

# Why semantics matter:

a demonstration on knowledge-based control system design

Wim Pessemier  
ICALEPCS 2015  
Melbourne

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their relations

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**

*plug\_X*

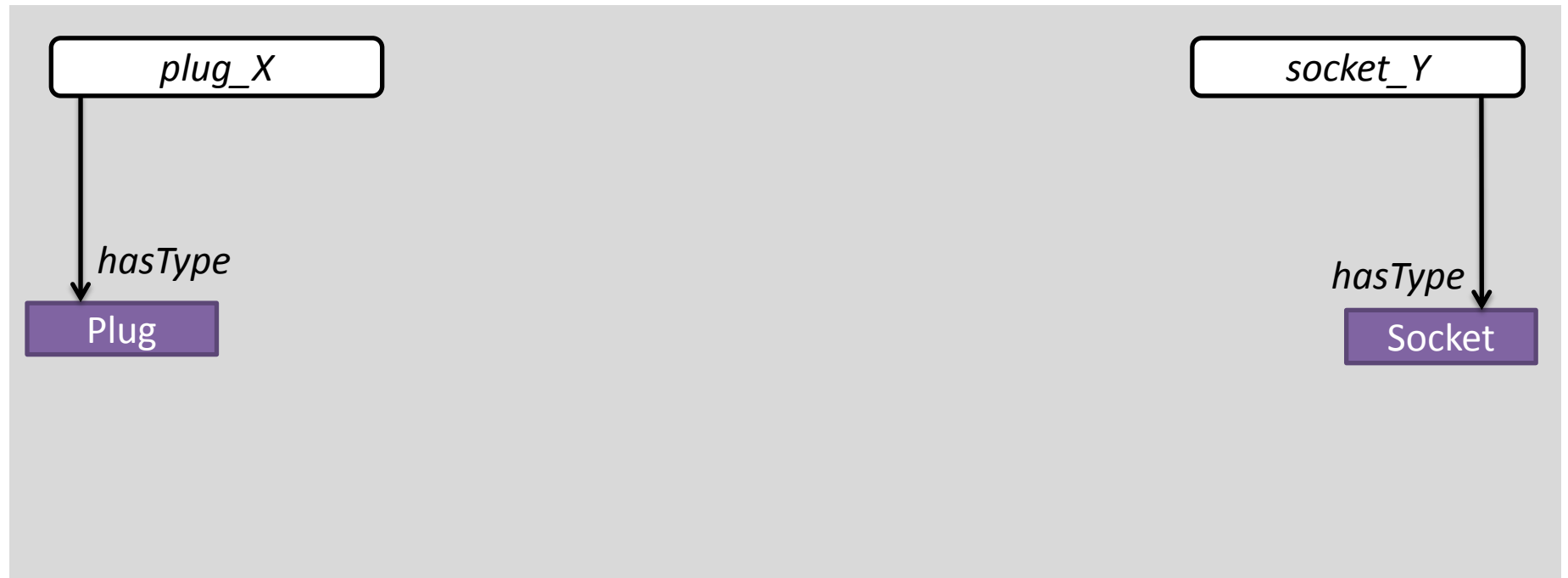
*socket\_Y*

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



GENERIC

*hasType*

ELECTRIC

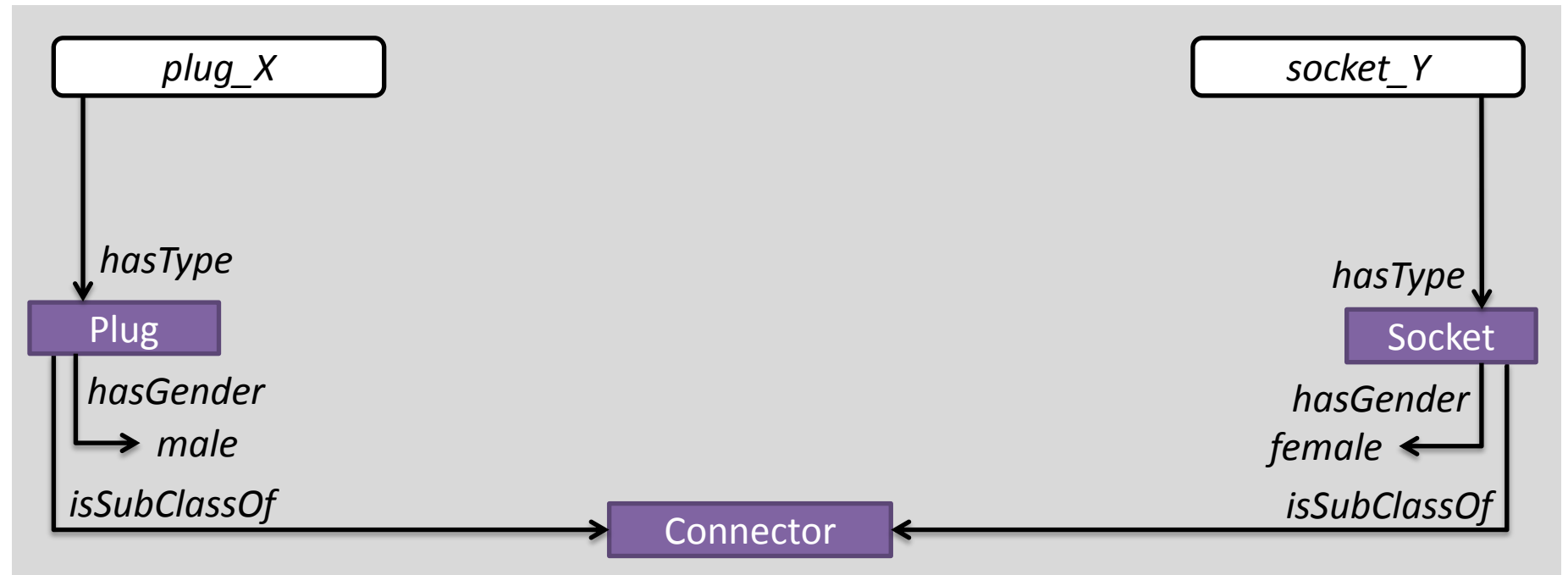
Plug  
Socket

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



GENERIC

hasType

ELECTRIC

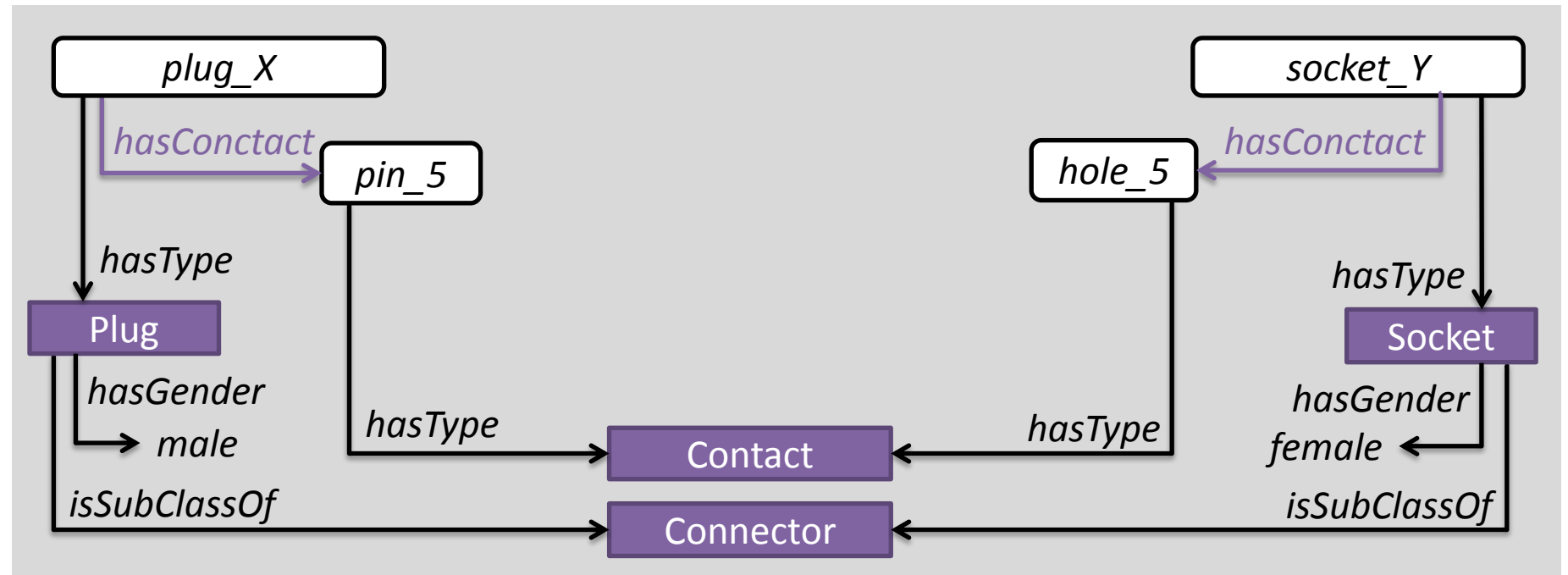
Plug  
Socket

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



### GENERIC

<i>hasType</i>	<i>isSubClassOf</i>
<i>hasGender</i>	

### ELECTRIC

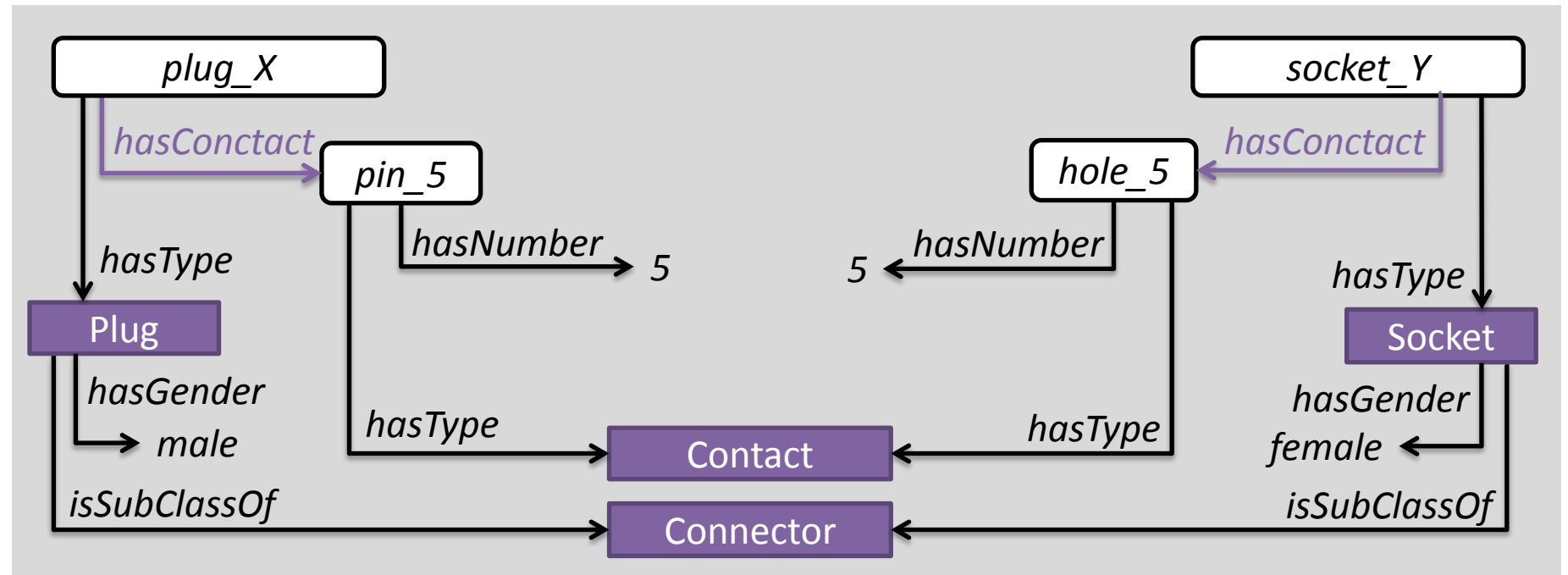
Plug	Connector
Socket	<i>hasContact</i>

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



### GENERIC

hasType	isSubClassOf
hasGender	hasNumber

### ELECTRIC

Plug	Connector	Contact
Socket	hasContact	

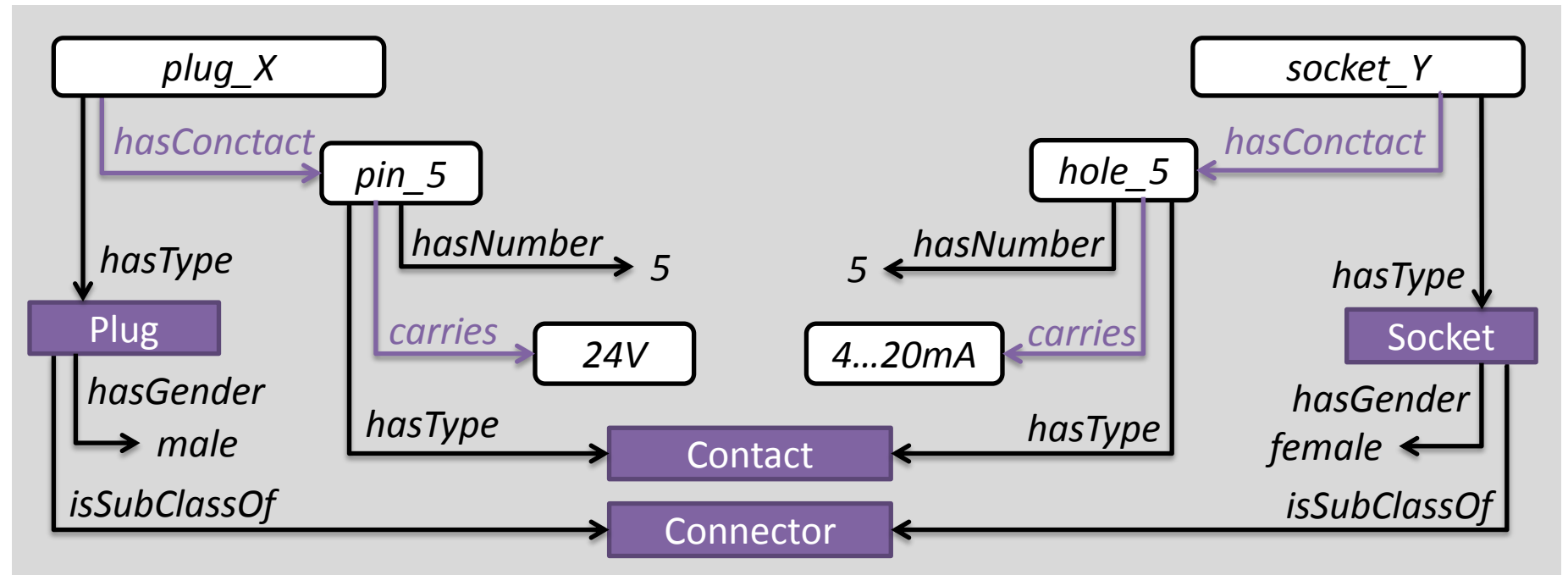


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



### GENERIC

hasType	isSubClassOf
hasGender	hasNumber

### ELECTRIC

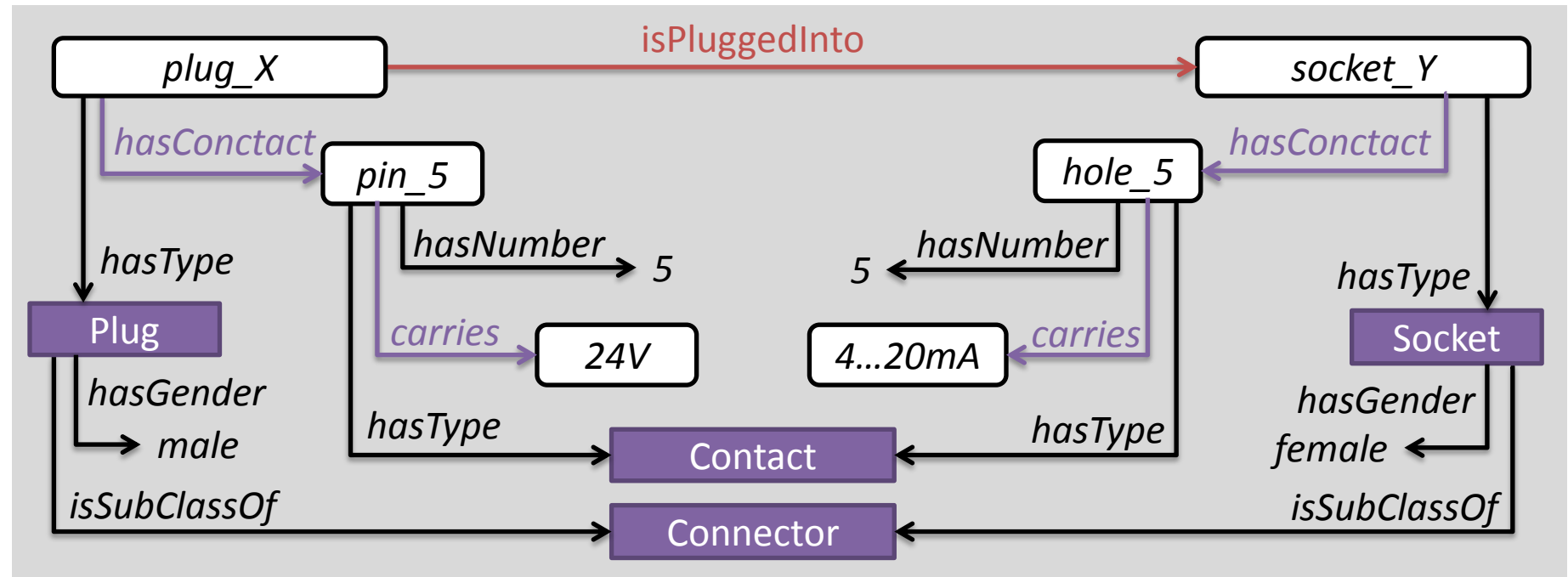
Plug	Connector	Contact
Socket	hasContact	carries

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



### GENERIC

hasType	isSubClassOf
hasGender	hasNumber

### ELECTRIC

Plug	Connector	Contact
Socket	hasContact	carries

### MECHANIC

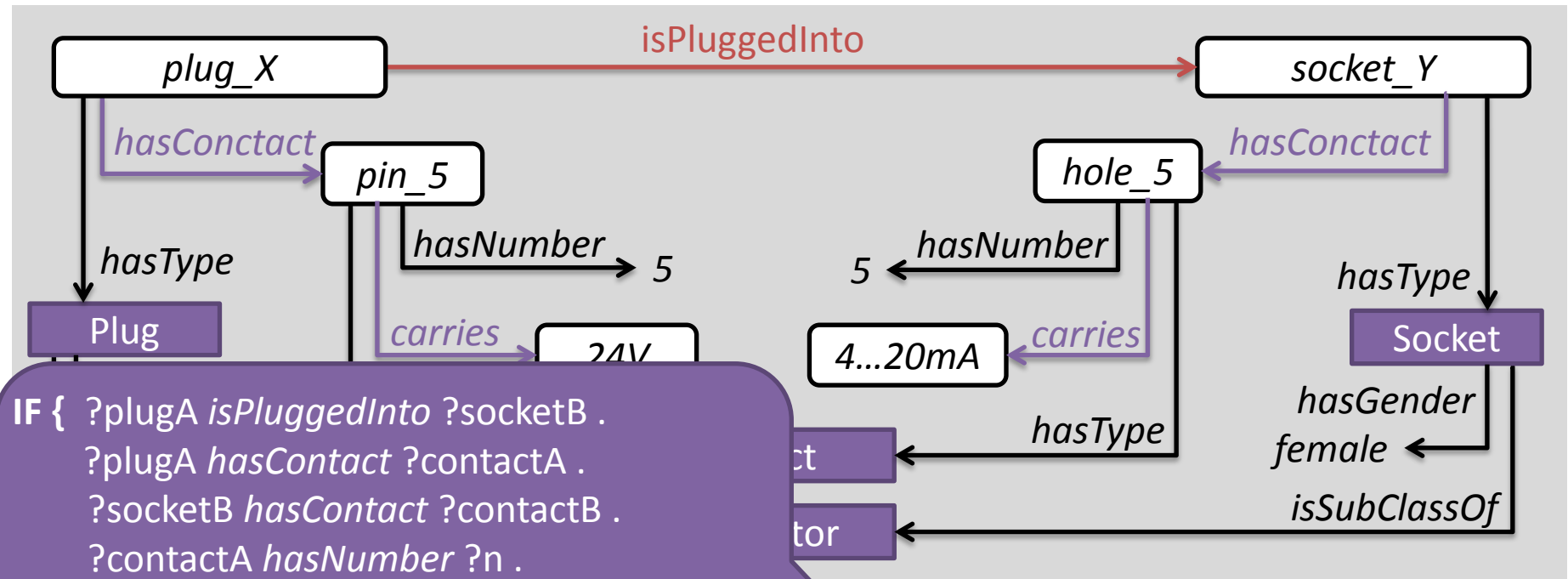
isPluggedInto
---------------

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



IF { ?plugA isPluggedInto ?socketB .  
?plugA hasContact ?contactA .  
?socketB hasContact ?contactB .  
?contactA hasNumber ?n .  
?contactB hasNumber ?n }  
THEN { ?contactA isConnectedTo ?contactB }

hasType	isSubClassOf	Plug	Connector	Contact
hasGender	hasNumber	Socket	hasContact	carries

MECHANIC

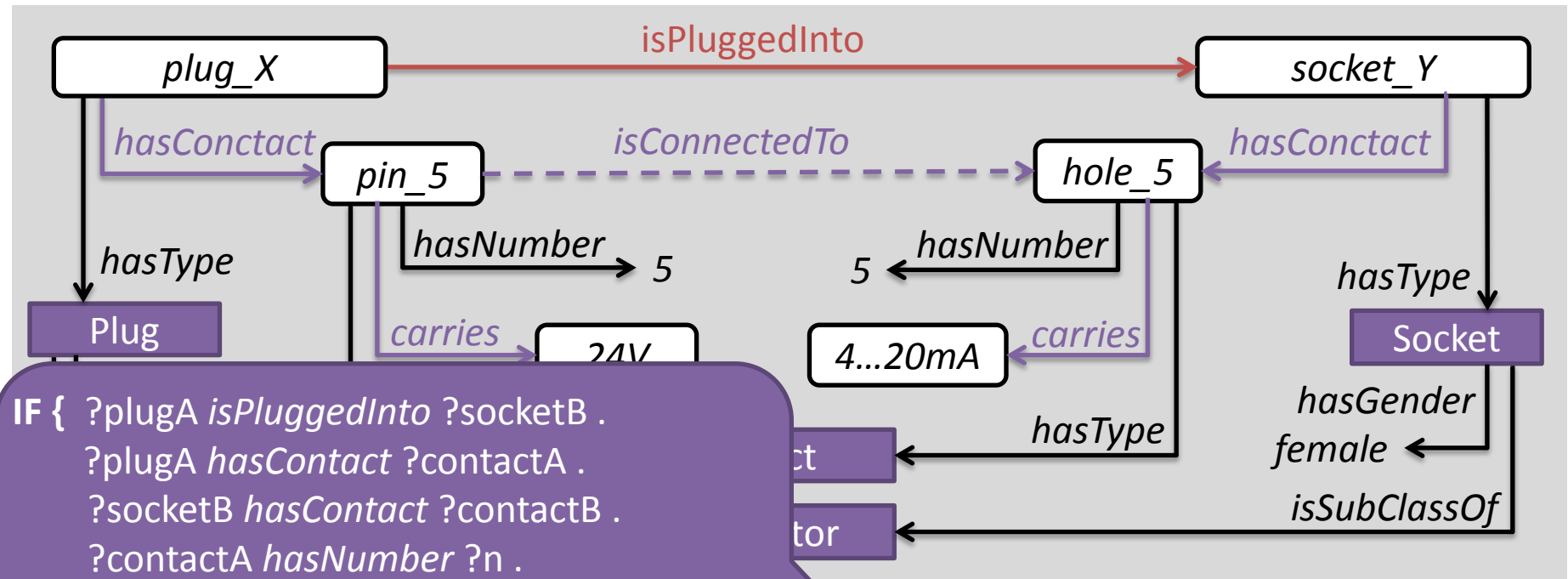
isPluggedInto

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



IF { ?plugA isPluggedInto ?socketB .  
?plugA hasContact ?contactA .  
?socketB hasContact ?contactB .  
?contactA hasNumber ?n .  
?contactB hasNumber ?n }  
THEN { ?contactA isConnectedTo ?contactB }

hasType	isSubClassOf	Plug	Connector	Contact
hasGender	hasNumber	Socket	hasContact	carries

MECHANIC

isPluggedInto

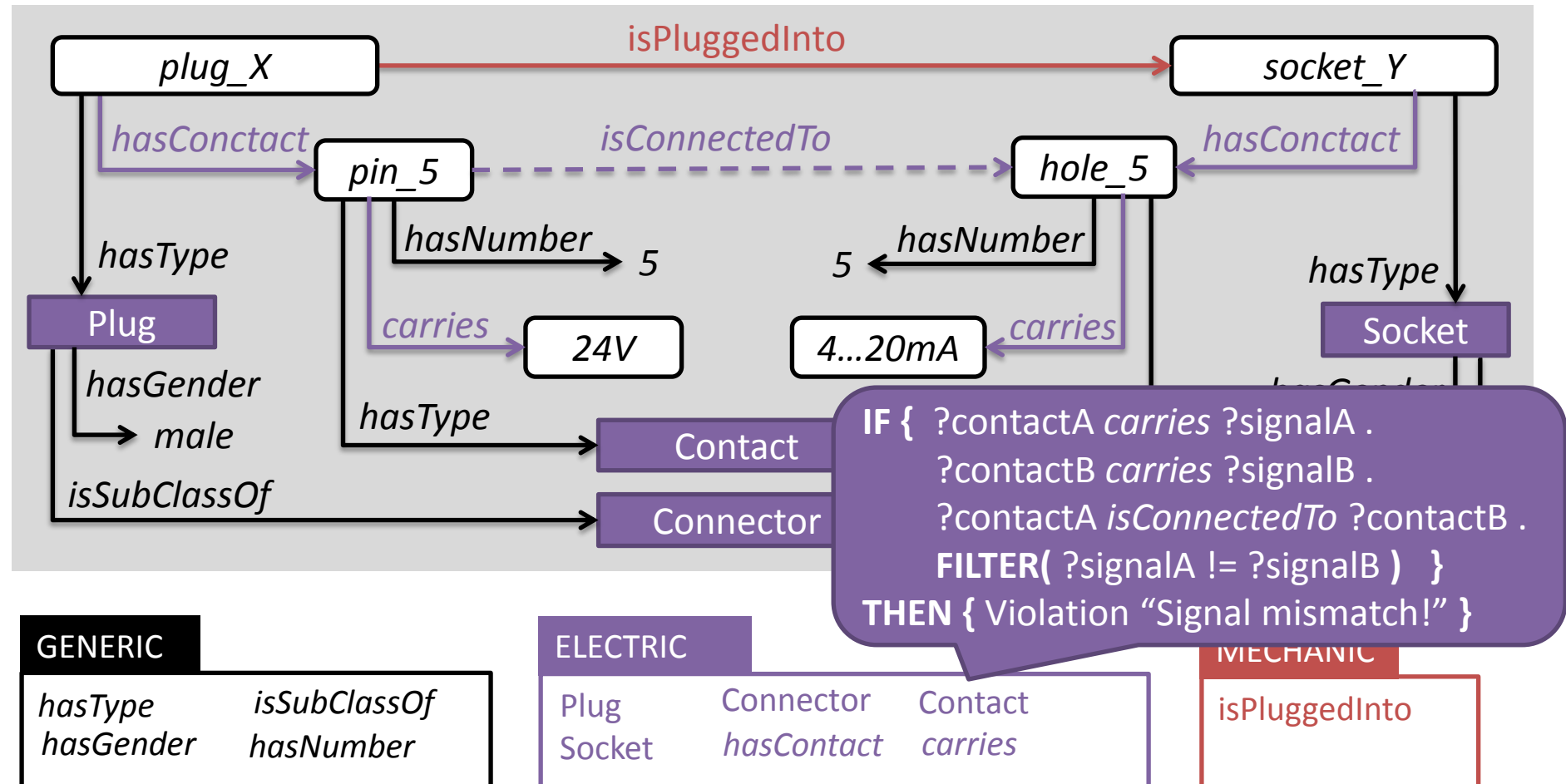


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



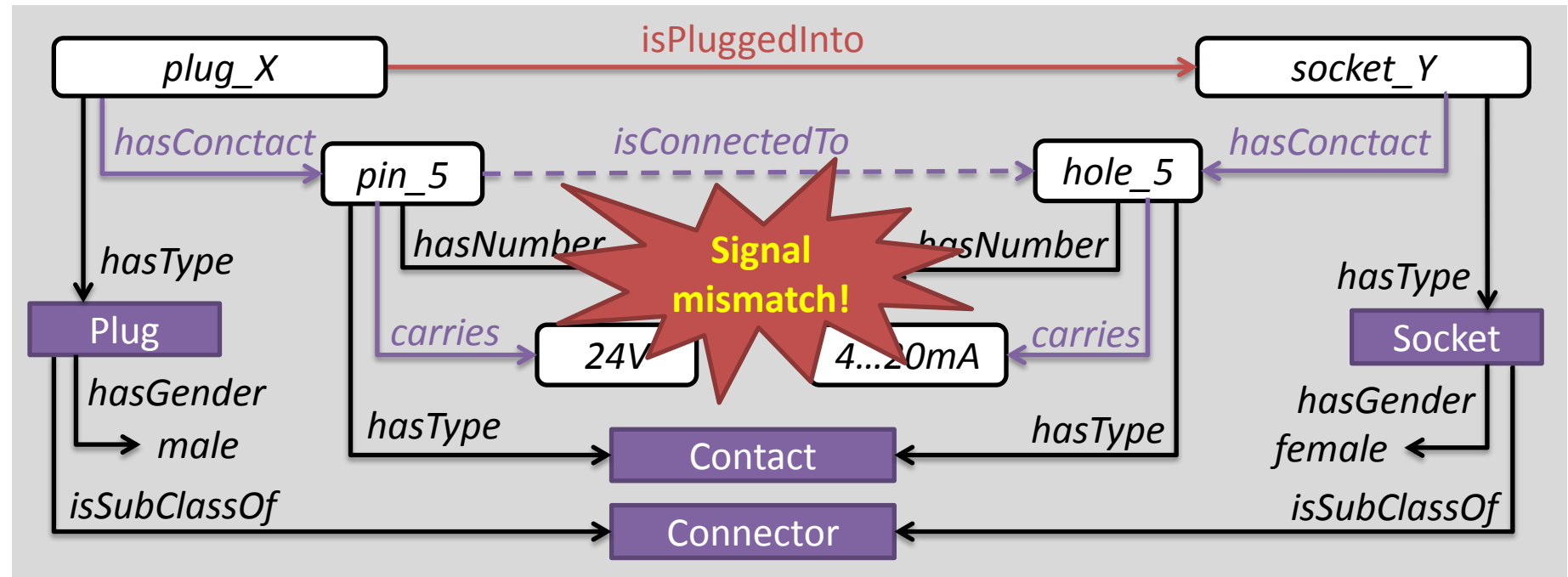


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

## What are semantic models?

- Models that describe
  - pieces of information (data, descriptions)
  - their **relations** → **meaning (semantics)**



### GENERIC

*hasType*      *isSubClassOf*  
*hasGender*    *hasNumber*

### ELECTRIC

Plug      Connector      Contact  
Socket    *hasContact*    *carries*

### MECHANIC

*isPluggedInto*

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

## Where to apply them?

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

## Where to apply them?



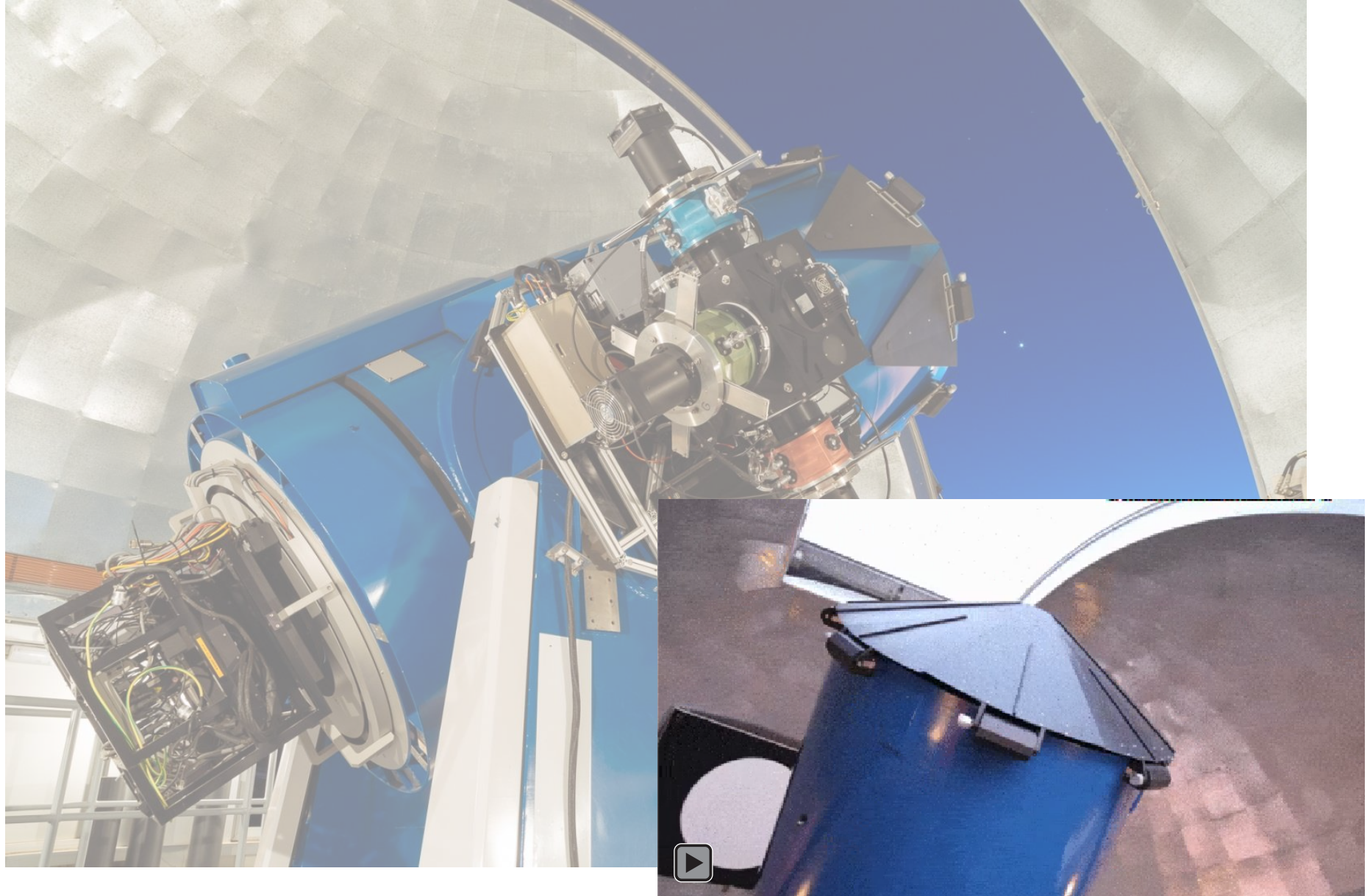
image: Péter Papics



## Why semantics matter?

- What are semantic models?
- **Where to apply them?**
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

## Where to apply them?



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

## How to apply them?

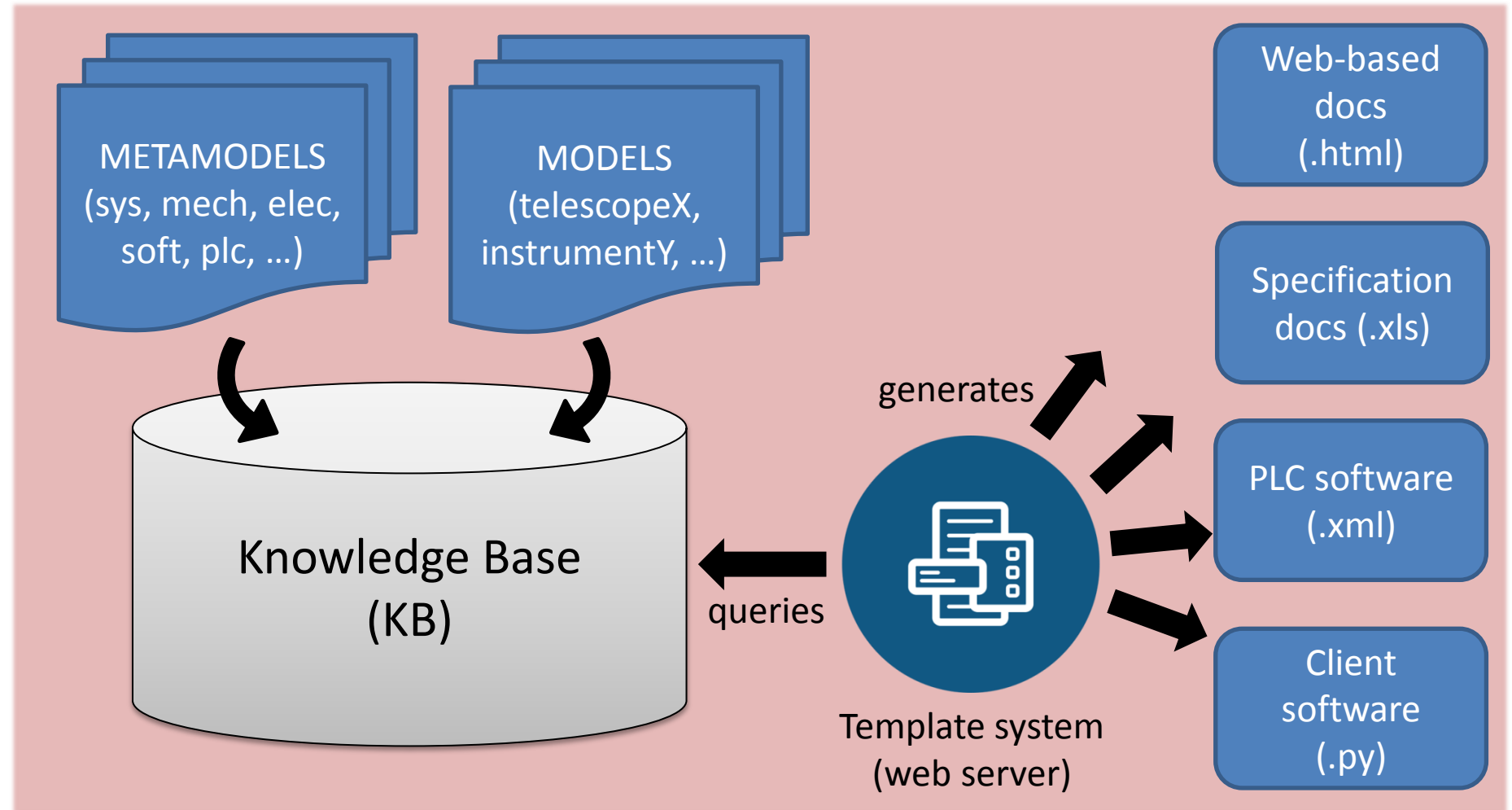


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- **How to apply them?**
- How to build them?
- How to use them?
- Conclusions

## How to apply them?

- Put them in a Knowledge Base and extract information!



**OntoManager**

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

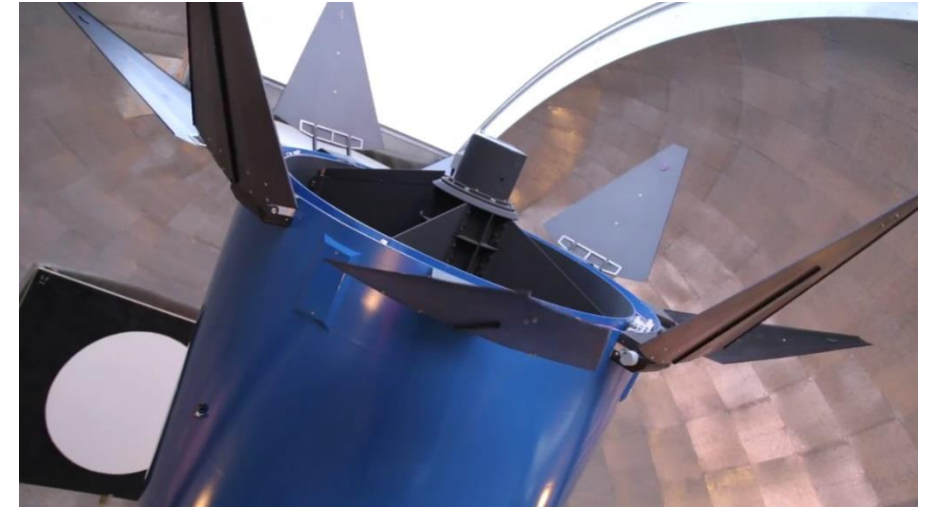
## How to build them?

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Using an existing modeling language?
  - UML, SysML, ... : semantics not sufficiently formal
  - Modeling languages have no “programming” capabilities (loops, functions, if-then, ...)



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Using an existing modeling language?

- UML, SysML, ... : semantics not sufficiently formal
- Modeling languages have no “programming” capabilities (loops, functions, if-then, ...)



- Using a Domain Specific Language (DSL)?

- Internal DSL called Ontoscript
- Based on coffeescript (~javascript)
- Idea “adopted” from the Giant Magellan Telescope project [1]

[1] J. M. Filgueira, “GMT software and controls overview”, Proc. SPIE 8451, Amsterdam, July 2012, 845111

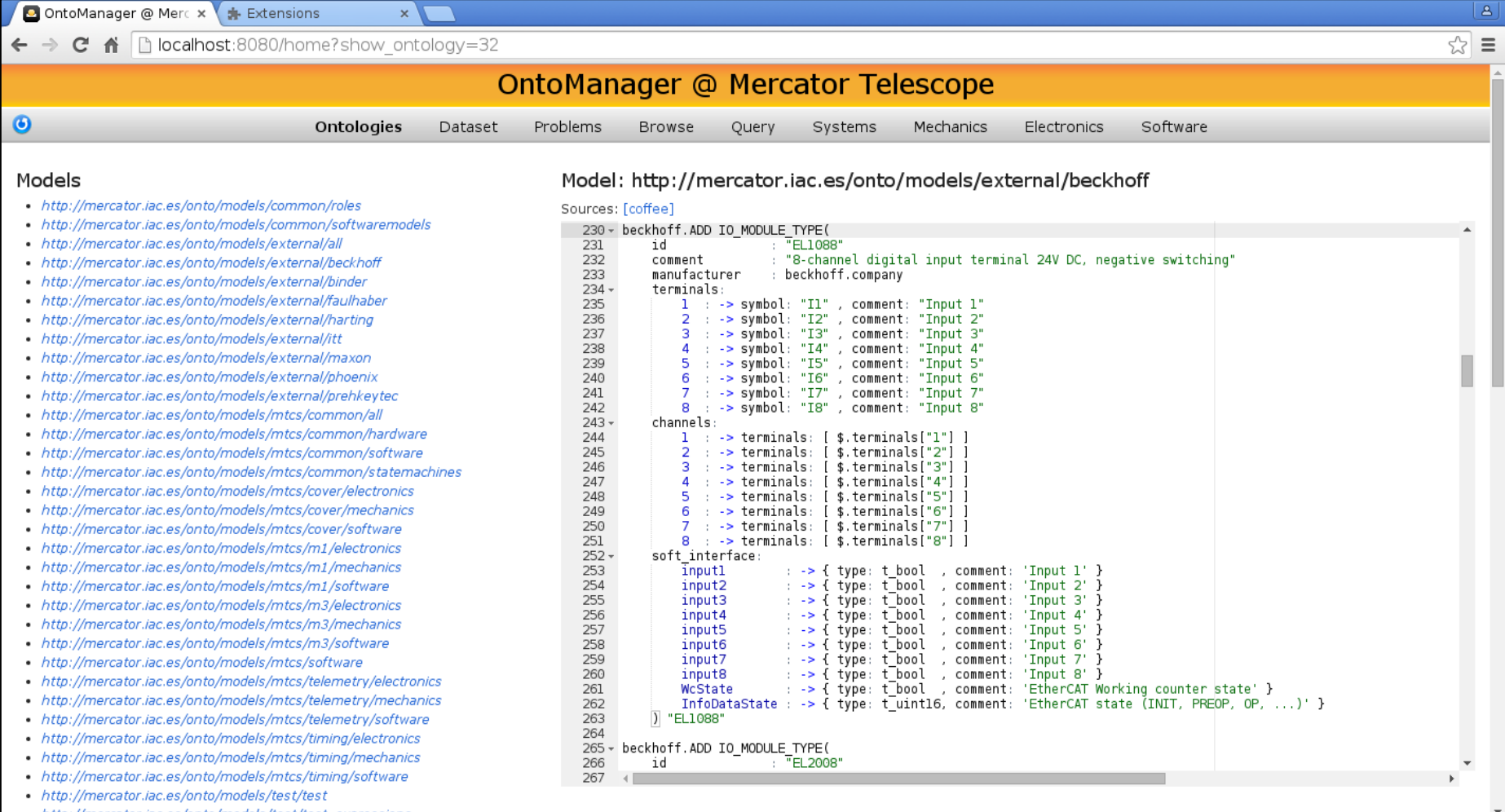


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Example: model of an I/O module **type**



The screenshot shows the OntoManager @ Mercator Telescope web application. The browser address bar displays `localhost:8080/home?show_ontology=32`. The application title is "OntoManager @ Mercator Telescope". The main navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology URLs, including `http://mercator.iac.es/onto/models/common/roles`, `http://mercator.iac.es/onto/models/common/softwaremodels`, `http://mercator.iac.es/onto/models/external/all`, `http://mercator.iac.es/onto/models/external/beckhoff`, `http://mercator.iac.es/onto/models/external/binder`, `http://mercator.iac.es/onto/models/external/faulhaber`, `http://mercator.iac.es/onto/models/external/harting`, `http://mercator.iac.es/onto/models/external/itt`, `http://mercator.iac.es/onto/models/external/maxon`, `http://mercator.iac.es/onto/models/external/phoenix`, `http://mercator.iac.es/onto/models/external/prehkeytec`, `http://mercator.iac.es/onto/models/mtcs/common/all`, `http://mercator.iac.es/onto/models/mtcs/common/hardware`, `http://mercator.iac.es/onto/models/mtcs/common/software`, `http://mercator.iac.es/onto/models/mtcs/common/statemachines`, `http://mercator.iac.es/onto/models/mtcs/cover/electronics`, `http://mercator.iac.es/onto/models/mtcs/cover/mechanics`, `http://mercator.iac.es/onto/models/mtcs/cover/software`, `http://mercator.iac.es/onto/models/mtcs/m1/electronics`, `http://mercator.iac.es/onto/models/mtcs/m1/mechanics`, `http://mercator.iac.es/onto/models/mtcs/m1/software`, `http://mercator.iac.es/onto/models/mtcs/m3/electronics`, `http://mercator.iac.es/onto/models/mtcs/m3/mechanics`, `http://mercator.iac.es/onto/models/mtcs/m3/software`, `http://mercator.iac.es/onto/models/mtcs/telemetry/electronics`, `http://mercator.iac.es/onto/models/mtcs/telemetry/mechanics`, `http://mercator.iac.es/onto/models/mtcs/telemetry/software`, `http://mercator.iac.es/onto/models/mtcs/timing/electronics`, `http://mercator.iac.es/onto/models/mtcs/timing/mechanics`, `http://mercator.iac.es/onto/models/mtcs/timing/software`, `http://mercator.iac.es/onto/models/test/test`, and `http://mercator.iac.es/onto/models/test/test_expressions`.

The main content area displays the model for `http://mercator.iac.es/onto/models/external/beckhoff`. The "Sources" section lists `[coffee]`. The model is defined in the following code block:

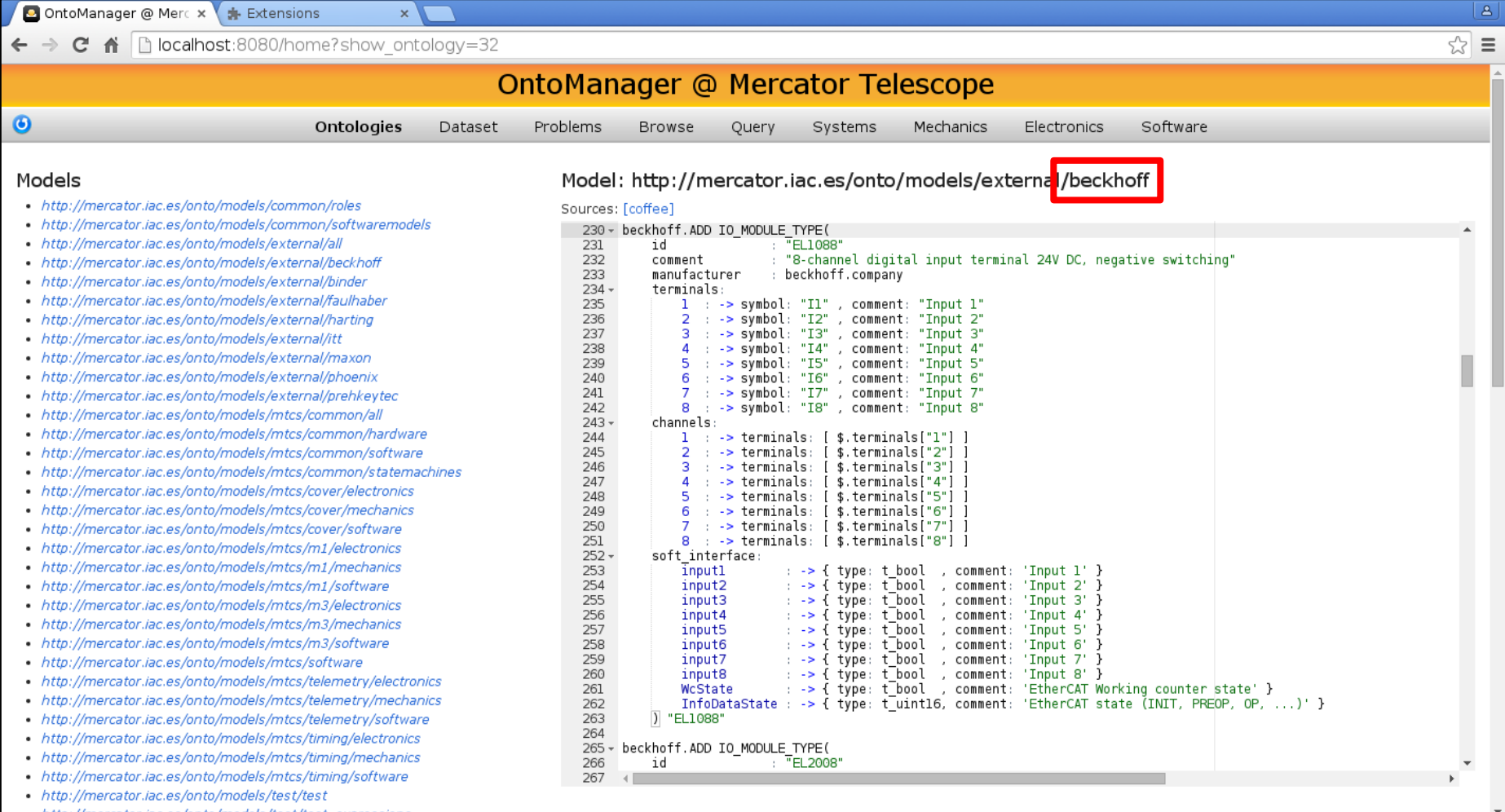
```
230 beckhoff.ADD IO_MODULE_TYPE(  
231   id      : "EL1088"  
232   comment : "8-channel digital input terminal 24V DC, negative switching"  
233   manufacturer : bechhoff.company  
234   terminals:  
235     1 : -> symbol: "I1", comment: "Input 1"  
236     2 : -> symbol: "I2", comment: "Input 2"  
237     3 : -> symbol: "I3", comment: "Input 3"  
238     4 : -> symbol: "I4", comment: "Input 4"  
239     5 : -> symbol: "I5", comment: "Input 5"  
240     6 : -> symbol: "I6", comment: "Input 6"  
241     7 : -> symbol: "I7", comment: "Input 7"  
242     8 : -> symbol: "I8", comment: "Input 8"  
243   channels:  
244     1 : -> terminals: [ $.terminals["1"] ]  
245     2 : -> terminals: [ $.terminals["2"] ]  
246     3 : -> terminals: [ $.terminals["3"] ]  
247     4 : -> terminals: [ $.terminals["4"] ]  
248     5 : -> terminals: [ $.terminals["5"] ]  
249     6 : -> terminals: [ $.terminals["6"] ]  
250     7 : -> terminals: [ $.terminals["7"] ]  
251     8 : -> terminals: [ $.terminals["8"] ]  
252   soft_interface:  
253     input1 : -> { type: t_bool , comment: 'Input 1' }  
254     input2 : -> { type: t_bool , comment: 'Input 2' }  
255     input3 : -> { type: t_bool , comment: 'Input 3' }  
256     input4 : -> { type: t_bool , comment: 'Input 4' }  
257     input5 : -> { type: t_bool , comment: 'Input 5' }  
258     input6 : -> { type: t_bool , comment: 'Input 6' }  
259     input7 : -> { type: t_bool , comment: 'Input 7' }  
260     input8 : -> { type: t_bool , comment: 'Input 8' }  
261     WcState : -> { type: t_bool , comment: 'EtherCAT Working counter state' }  
262     InfoDataState : -> { type: t_uint16, comment: 'EtherCAT state (INIT, PREOP, OP, ...)' }  
263   ) "EL1088"  
264  
265 beckhoff.ADD IO_MODULE_TYPE(  
266   id      : "EL2008"  
267
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Example: model of an I/O module **type**



The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar displays `localhost:8080/home?show_ontology=32`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology URLs, including `http://mercator.iac.es/onto/models/external/beckhoff`. The main content area displays the model for `http://mercator.iac.es/onto/models/external/beckhoff`, which is highlighted with a red box. The model is a CoffeeScript file defining an I/O module type.

```
Model: http://mercator.iac.es/onto/models/external/beckhoff
Sources: [coffee]
230 beckhoff.ADD IO_MODULE_TYPE(
231   id      : "EL1088"
232   comment : "8-channel digital input terminal 24V DC, negative switching"
233   manufacturer : beckhoff.company
234   terminals:
235     1 : -> symbol: "I1", comment: "Input 1"
236     2 : -> symbol: "I2", comment: "Input 2"
237     3 : -> symbol: "I3", comment: "Input 3"
238     4 : -> symbol: "I4", comment: "Input 4"
239     5 : -> symbol: "I5", comment: "Input 5"
240     6 : -> symbol: "I6", comment: "Input 6"
241     7 : -> symbol: "I7", comment: "Input 7"
242     8 : -> symbol: "I8", comment: "Input 8"
243   channels:
244     1 : -> terminals: [ $.terminals["1"] ]
245     2 : -> terminals: [ $.terminals["2"] ]
246     3 : -> terminals: [ $.terminals["3"] ]
247     4 : -> terminals: [ $.terminals["4"] ]
248     5 : -> terminals: [ $.terminals["5"] ]
249     6 : -> terminals: [ $.terminals["6"] ]
250     7 : -> terminals: [ $.terminals["7"] ]
251     8 : -> terminals: [ $.terminals["8"] ]
252   soft_interface:
253     input1 : -> { type: t_bool , comment: 'Input 1' }
254     input2 : -> { type: t_bool , comment: 'Input 2' }
255     input3 : -> { type: t_bool , comment: 'Input 3' }
256     input4 : -> { type: t_bool , comment: 'Input 4' }
257     input5 : -> { type: t_bool , comment: 'Input 5' }
258     input6 : -> { type: t_bool , comment: 'Input 6' }
259     input7 : -> { type: t_bool , comment: 'Input 7' }
260     input8 : -> { type: t_bool , comment: 'Input 8' }
261     WcState : -> { type: t_bool , comment: 'EtherCAT Working counter state' }
262     InfoDataState : -> { type: t_uint16, comment: 'EtherCAT state (INIT, PREOP, OP, ...)' }
263   ) "EL1088"
264
265 beckhoff.ADD IO_MODULE_TYPE(
266   id      : "EL2008"
267
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Example: model of an I/O module **type**

The screenshot displays the OntoManager @ Mercator Telescope web interface. On the left, a list of ontologies is shown, including various roles, software models, and specific modules like EL1008 and EL2008. A photograph of the EL1008 I/O module is overlaid on this list. The main panel on the right shows the definition of the 'beckhoff' model, which is highlighted with a red box. The definition is written in a CoffeeScript-like syntax and describes an 8-channel digital input terminal with 24V DC, negative switching. It lists 8 terminals, each with a symbol (I1-I8) and comment (Input 1-8). It also defines 8 channels, each mapping to a terminal. Finally, it defines a soft interface with 8 boolean inputs (input1-input8) and two state variables: WcState (EtherCAT Working counter state) and InfoDataState (EtherCAT state).

Model: <http://mercator.iac.es/onto/models/external/beckhoff>

```
beckhoff.ADD IO_MODULE_TYPE(  
  id      : "EL1088"  
  comment : "8-channel digital input terminal 24V DC, negative switching"  
  manufacturer : beckhoff.company  
  terminals:  
    1 : -> symbol: "I1" , comment: "Input 1"  
    2 : -> symbol: "I2" , comment: "Input 2"  
    3 : -> symbol: "I3" , comment: "Input 3"  
    4 : -> symbol: "I4" , comment: "Input 4"  
    5 : -> symbol: "I5" , comment: "Input 5"  
    6 : -> symbol: "I6" , comment: "Input 6"  
    7 : -> symbol: "I7" , comment: "Input 7"  
    8 : -> symbol: "I8" , comment: "Input 8"  
  channels:  
    1 : -> terminals: [ $.terminals["1"] ]  
    2 : -> terminals: [ $.terminals["2"] ]  
    3 : -> terminals: [ $.terminals["3"] ]  
    4 : -> terminals: [ $.terminals["4"] ]  
    5 : -> terminals: [ $.terminals["5"] ]  
    6 : -> terminals: [ $.terminals["6"] ]  
    7 : -> terminals: [ $.terminals["7"] ]  
    8 : -> terminals: [ $.terminals["8"] ]  
  soft_interface:  
    input1 : -> { type: t_bool , comment: 'Input 1' }  
    input2 : -> { type: t_bool , comment: 'Input 2' }  
    input3 : -> { type: t_bool , comment: 'Input 3' }  
    input4 : -> { type: t_bool , comment: 'Input 4' }  
    input5 : -> { type: t_bool , comment: 'Input 5' }  
    input6 : -> { type: t_bool , comment: 'Input 6' }  
    input7 : -> { type: t_bool , comment: 'Input 7' }  
    input8 : -> { type: t_bool , comment: 'Input 8' }  
    WcState : -> { type: t_bool , comment: 'EtherCAT Working counter state' }  
    InfoDataState : -> { type: t_uint16, comment: 'EtherCAT state (INIT, PREOP, OP, ...)' }  
)  
"EL1088"
```

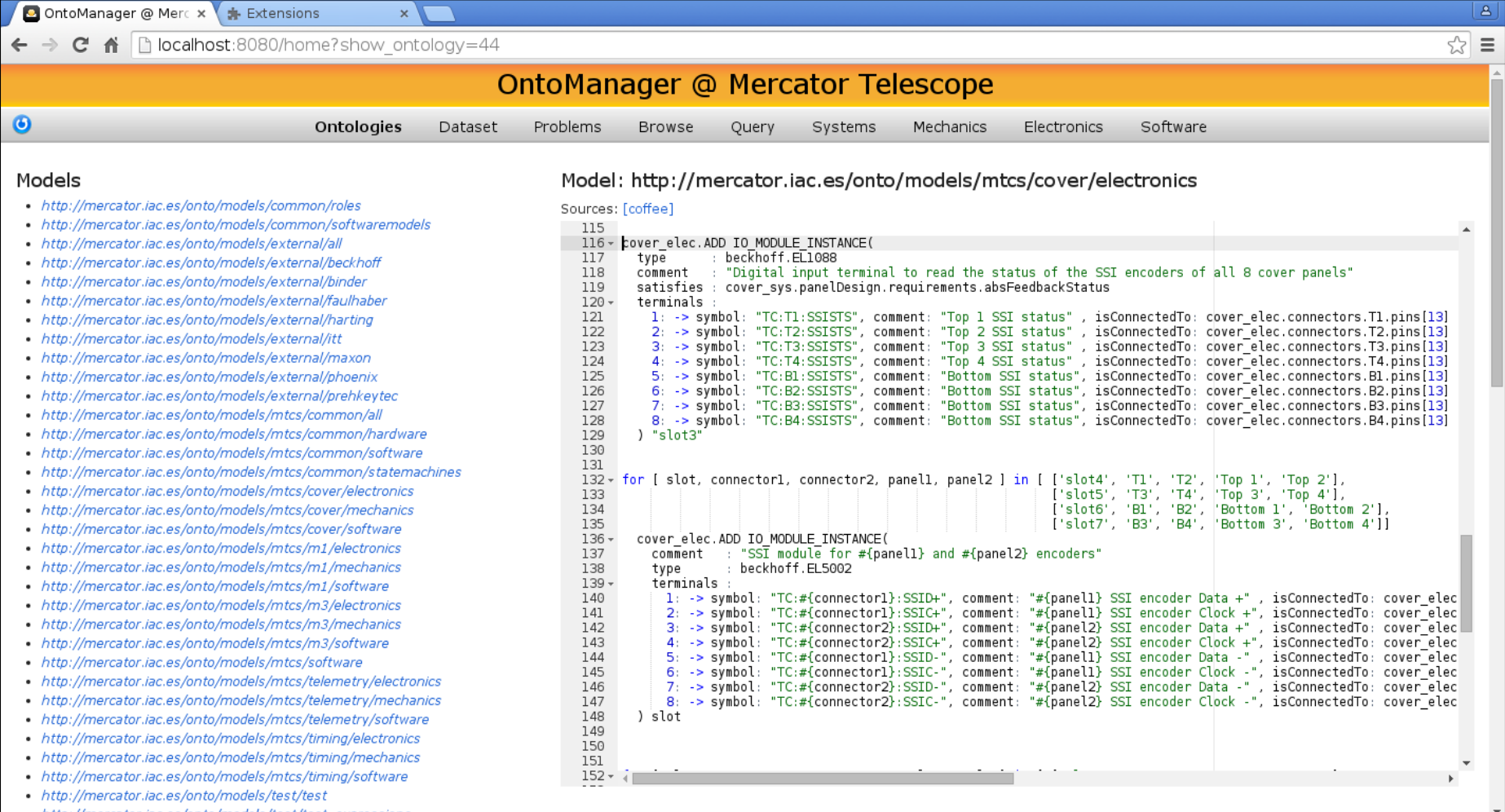


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Example: model of an I/O module instance



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=44`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology URLs, including:

- <http://mercator.iac.es/onto/models/common/roles>
- <http://mercator.iac.es/onto/models/common/softwaremodels>
- <http://mercator.iac.es/onto/models/external/all>
- <http://mercator.iac.es/onto/models/external/beckhoff>
- <http://mercator.iac.es/onto/models/external/binder>
- <http://mercator.iac.es/onto/models/external/faulhaber>
- <http://mercator.iac.es/onto/models/external/harting>
- <http://mercator.iac.es/onto/models/external/itt>
- <http://mercator.iac.es/onto/models/external/maxon>
- <http://mercator.iac.es/onto/models/external/phoenix>
- <http://mercator.iac.es/onto/models/external/prehkeytec>
- <http://mercator.iac.es/onto/models/mtcs/common/all>
- <http://mercator.iac.es/onto/models/mtcs/common/hardware>
- <http://mercator.iac.es/onto/models/mtcs/common/software>
- <http://mercator.iac.es/onto/models/mtcs/common/statemachines>
- <http://mercator.iac.es/onto/models/mtcs/cover/electronics>
- <http://mercator.iac.es/onto/models/mtcs/cover/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/cover/software>
- <http://mercator.iac.es/onto/models/mtcs/m1/electronics>
- <http://mercator.iac.es/onto/models/mtcs/m1/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/m1/software>
- <http://mercator.iac.es/onto/models/mtcs/m3/electronics>
- <http://mercator.iac.es/onto/models/mtcs/m3/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/m3/software>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/electronics>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/telemetry/software>
- <http://mercator.iac.es/onto/models/mtcs/timing/electronics>
- <http://mercator.iac.es/onto/models/mtcs/timing/mechanics>
- <http://mercator.iac.es/onto/models/mtcs/timing/software>
- <http://mercator.iac.es/onto/models/test/test>
- [http://mercator.iac.es/onto/models/test/test\\_expressions](http://mercator.iac.es/onto/models/test/test_expressions)

The "Model: <http://mercator.iac.es/onto/models/mtcs/cover/electronics>" section shows the source code for the `cover_elec` module. The code defines the module's type, comment, and terminals. It includes a `for` loop to iterate over slots and connectors, and a `cover_elec.ADD IO_MODULE_INSTANCE` block to define the module's behavior.

```
115
116 cover_elec.ADD IO_MODULE_INSTANCE(
117   type      : beckhoff.EL1088
118   comment    : "Digital input terminal to read the status of the SSI encoders of all 8 cover panels"
119   satisfies  : cover_sys.panelDesign.requirements.absFeedbackStatus
120   terminals :
121     1: -> symbol: "TC:T1:SSISTS", comment: "Top 1 SSI status", isConnectedTo: cover_elec.connectors.T1.pins[13]
122     2: -> symbol: "TC:T2:SSISTS", comment: "Top 2 SSI status", isConnectedTo: cover_elec.connectors.T2.pins[13]
123     3: -> symbol: "TC:T3:SSISTS", comment: "Top 3 SSI status", isConnectedTo: cover_elec.connectors.T3.pins[13]
124     4: -> symbol: "TC:T4:SSISTS", comment: "Top 4 SSI status", isConnectedTo: cover_elec.connectors.T4.pins[13]
125     5: -> symbol: "TC:B1:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B1.pins[13]
126     6: -> symbol: "TC:B2:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B2.pins[13]
127     7: -> symbol: "TC:B3:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B3.pins[13]
128     8: -> symbol: "TC:B4:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B4.pins[13]
129   ) "slot3"
130
131
132 for [ slot, connector1, connector2, panel1, panel2 ] in [ ['slot4', 'T1', 'T2', 'Top 1', 'Top 2'],
133   ['slot5', 'T3', 'T4', 'Top 3', 'Top 4'],
134   ['slot6', 'B1', 'B2', 'Bottom 1', 'Bottom 2'],
135   ['slot7', 'B3', 'B4', 'Bottom 3', 'Bottom 4']]
136
137 cover_elec.ADD IO_MODULE_INSTANCE(
138   comment    : "SSI module for #{panel1} and #{panel2} encoders"
139   type      : beckhoff.EL5002
140   terminals :
141     1: -> symbol: "TC:#{connector1}:SSID+", comment: "#{panel1} SSI encoder Data +", isConnectedTo: cover_elec
142     2: -> symbol: "TC:#{connector1}:SSIC+", comment: "#{panel1} SSI encoder Clock +", isConnectedTo: cover_elec
143     3: -> symbol: "TC:#{connector2}:SSID+", comment: "#{panel2} SSI encoder Data +", isConnectedTo: cover_elec
144     4: -> symbol: "TC:#{connector2}:SSIC+", comment: "#{panel2} SSI encoder Clock +", isConnectedTo: cover_elec
145     5: -> symbol: "TC:#{connector1}:SSID-", comment: "#{panel1} SSI encoder Data -", isConnectedTo: cover_elec
146     6: -> symbol: "TC:#{connector1}:SSIC-", comment: "#{panel1} SSI encoder Clock -", isConnectedTo: cover_elec
147     7: -> symbol: "TC:#{connector2}:SSID-", comment: "#{panel2} SSI encoder Data -", isConnectedTo: cover_elec
148     8: -> symbol: "TC:#{connector2}:SSIC-", comment: "#{panel2} SSI encoder Clock -", isConnectedTo: cover_elec
149   ) slot
150
151
152
```

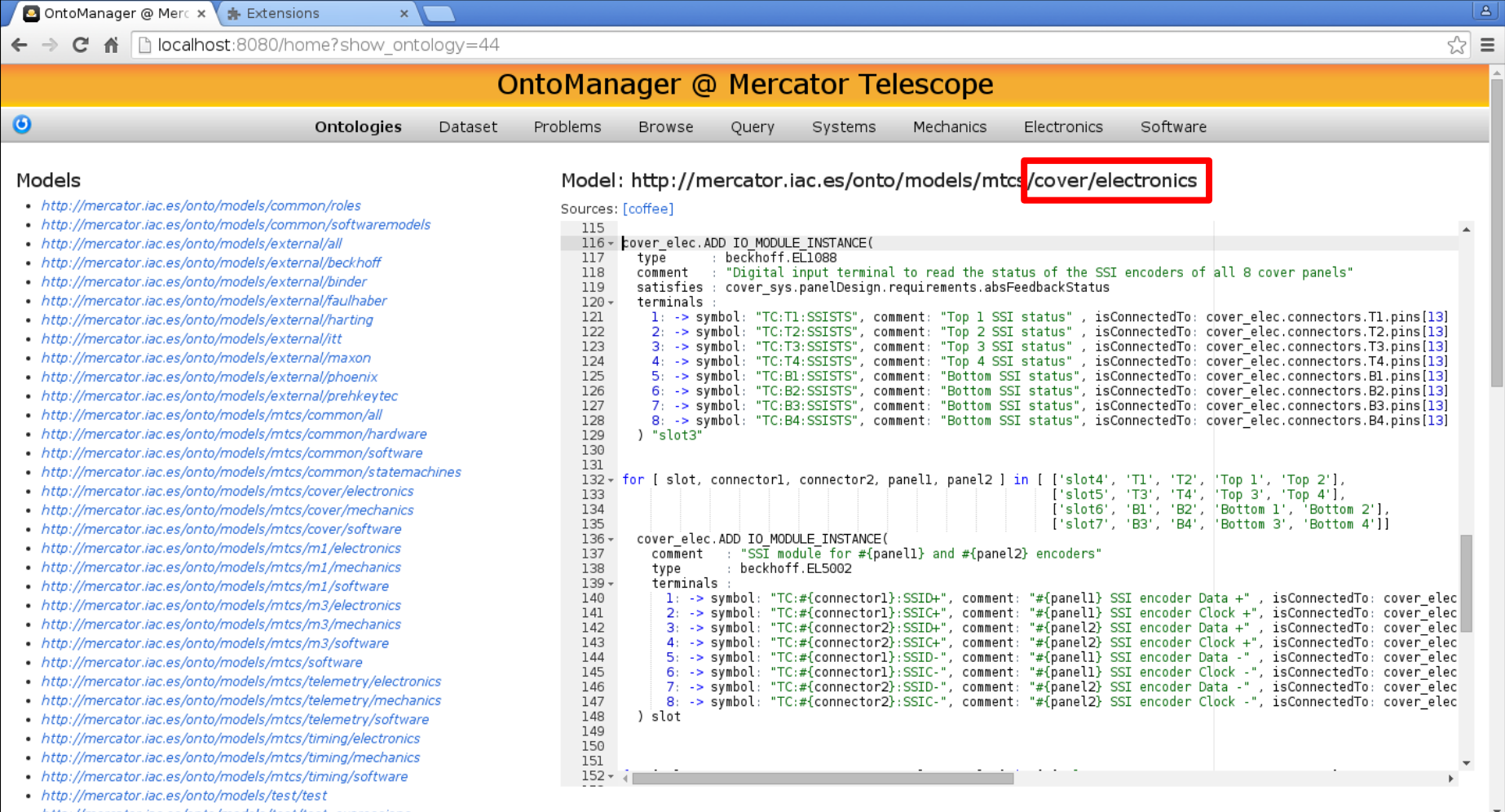


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Example: model of an I/O module **instance**



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows the URL `localhost:8080/home?show_ontology=44`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology models, including `http://mercator.iac.es/onto/models/mtcs/cover/electronics`. The main content area displays the model `http://mercator.iac.es/onto/models/mtcs/cover/electronics` (highlighted with a red box). The model is a list of instances, with the first instance being `cover_elec.ADD IO_MODULE_INSTANCE`. The instance details include:

- `type`: `beckhoff.EL1088`
- `comment`: "Digital input terminal to read the status of the SSI encoders of all 8 cover panels"
- `satisfies`: `cover_sys.panelDesign.requirements.absFeedbackStatus`
- `terminals`:
  - 1: `-> symbol: "TC:T1:SSISTS", comment: "Top 1 SSI status", isConnectedTo: cover_elec.connectors.T1.pins[13]`
  - 2: `-> symbol: "TC:T2:SSISTS", comment: "Top 2 SSI status", isConnectedTo: cover_elec.connectors.T2.pins[13]`
  - 3: `-> symbol: "TC:T3:SSISTS", comment: "Top 3 SSI status", isConnectedTo: cover_elec.connectors.T3.pins[13]`
  - 4: `-> symbol: "TC:T4:SSISTS", comment: "Top 4 SSI status", isConnectedTo: cover_elec.connectors.T4.pins[13]`
  - 5: `-> symbol: "TC:B1:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B1.pins[13]`
  - 6: `-> symbol: "TC:B2:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B2.pins[13]`
  - 7: `-> symbol: "TC:B3:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B3.pins[13]`
  - 8: `-> symbol: "TC:B4:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B4.pins[13]`

The second instance is `cover_elec.ADD IO_MODULE_INSTANCE` with the following details:

- `comment`: "SSI module for #{panel1} and #{panel2} encoders"
- `type`: `beckhoff.EL5002`
- `terminals`:
  - 1: `-> symbol: "TC:#{connector1}:SSID+", comment: "#{panel1} SSI encoder Data +", isConnectedTo: cover_elec`
  - 2: `-> symbol: "TC:#{connector1}:SSIC+", comment: "#{panel1} SSI encoder Clock +", isConnectedTo: cover_elec`
  - 3: `-> symbol: "TC:#{connector2}:SSID+", comment: "#{panel2} SSI encoder Data +", isConnectedTo: cover_elec`
  - 4: `-> symbol: "TC:#{connector2}:SSIC+", comment: "#{panel2} SSI encoder Clock +", isConnectedTo: cover_elec`
  - 5: `-> symbol: "TC:#{connector1}:SSID-", comment: "#{panel1} SSI encoder Data -", isConnectedTo: cover_elec`
  - 6: `-> symbol: "TC:#{connector1}:SSIC-", comment: "#{panel1} SSI encoder Clock -", isConnectedTo: cover_elec`
  - 7: `-> symbol: "TC:#{connector2}:SSID-", comment: "#{panel2} SSI encoder Data -", isConnectedTo: cover_elec`
  - 8: `-> symbol: "TC:#{connector2}:SSIC-", comment: "#{panel2} SSI encoder Clock -", isConnectedTo: cover_elec`

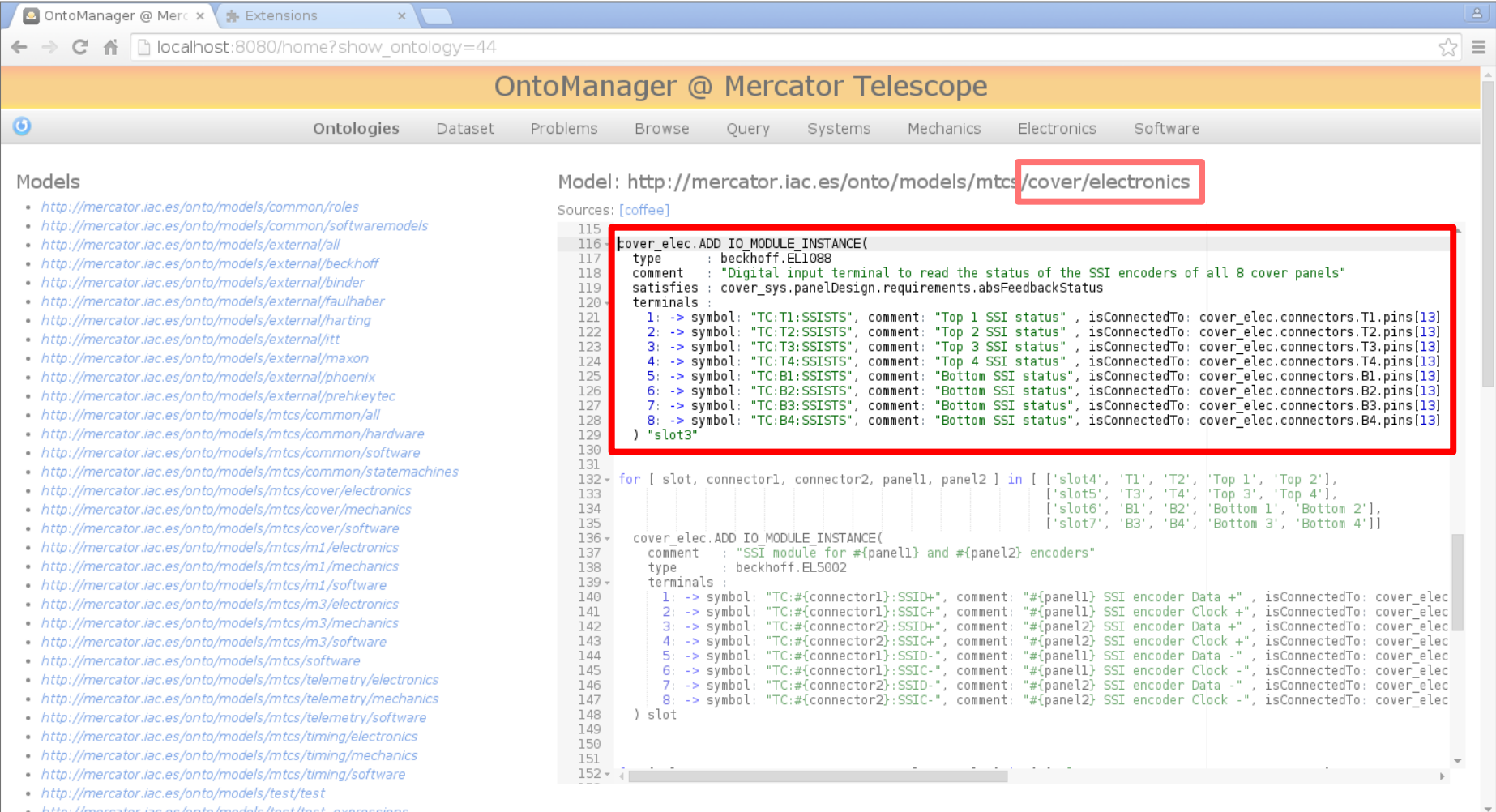
The instance is associated with the `slot` property.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- **How to build them?**
- How to use them?
- Conclusions

## How to build them?

- Example: model of an I/O module **instance**



The screenshot displays the OntoManager @ Mercator Telescope web interface. The browser address bar shows `localhost:8080/home?show_ontology=44`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software".

The "Models" section on the left lists various ontology models, including `http://mercator.iac.es/onto/models/mtcs/cover/electronics`. The main content area displays the model `Model: http://mercator.iac.es/onto/models/mtcs/cover/electronics` with sources `[coffee]`.

The model content is a Prolog-like code snippet defining an I/O module instance for a cover panel. The code is as follows:

```
cover_elec.ADD IO_MODULE_INSTANCE(  
  type      : beckhoff.EL1088  
  comment   : "Digital input terminal to read the status of the SSI encoders of all 8 cover panels"  
  satisfies : cover_sys.panelDesign.requirements.absFeedbackStatus  
  terminals :  
    1: -> symbol: "TC:T1:SSISTS", comment: "Top 1 SSI status", isConnectedTo: cover_elec.connectors.T1.pins[13]  
    2: -> symbol: "TC:T2:SSISTS", comment: "Top 2 SSI status", isConnectedTo: cover_elec.connectors.T2.pins[13]  
    3: -> symbol: "TC:T3:SSISTS", comment: "Top 3 SSI status", isConnectedTo: cover_elec.connectors.T3.pins[13]  
    4: -> symbol: "TC:T4:SSISTS", comment: "Top 4 SSI status", isConnectedTo: cover_elec.connectors.T4.pins[13]  
    5: -> symbol: "TC:B1:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B1.pins[13]  
    6: -> symbol: "TC:B2:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B2.pins[13]  
    7: -> symbol: "TC:B3:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B3.pins[13]  
    8: -> symbol: "TC:B4:SSISTS", comment: "Bottom SSI status", isConnectedTo: cover_elec.connectors.B4.pins[13]  
  ) "slot3"  
  
for [ slot, connector1, connector2, panel1, panel2 ] in [ ['slot4', 'T1', 'T2', 'Top 1', 'Top 2'],  
  ['slot5', 'T3', 'T4', 'Top 3', 'Top 4'],  
  ['slot6', 'B1', 'B2', 'Bottom 1', 'Bottom 2'],  
  ['slot7', 'B3', 'B4', 'Bottom 3', 'Bottom 4']]  
  cover_elec.ADD IO_MODULE_INSTANCE(  
    comment : "SSI module for #{panel1} and #{panel2} encoders"  
    type    : beckhoff.EL5002  
    terminals :  
      1: -> symbol: "TC:#{connector1}:SSID+", comment: "#{panel1} SSI encoder Data +", isConnectedTo: cover_elec  
      2: -> symbol: "TC:#{connector1}:SSIC+", comment: "#{panel1} SSI encoder Clock +", isConnectedTo: cover_elec  
      3: -> symbol: "TC:#{connector2}:SSID+", comment: "#{panel2} SSI encoder Data +", isConnectedTo: cover_elec  
      4: -> symbol: "TC:#{connector2}:SSIC+", comment: "#{panel2} SSI encoder Clock +", isConnectedTo: cover_elec  
      5: -> symbol: "TC:#{connector1}:SSID-", comment: "#{panel1} SSI encoder Data -", isConnectedTo: cover_elec  
      6: -> symbol: "TC:#{connector1}:SSIC-", comment: "#{panel1} SSI encoder Clock -", isConnectedTo: cover_elec  
      7: -> symbol: "TC:#{connector2}:SSID-", comment: "#{panel2} SSI encoder Data -", isConnectedTo: cover_elec  
      8: -> symbol: "TC:#{connector2}:SSIC-", comment: "#{panel2} SSI encoder Clock -", isConnectedTo: cover_elec  
    ) slot
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

## How to use them?

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design





## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

The screenshot shows a web browser window with the title "OntoManager @ Merc" and a single tab. The address bar displays the URL "localhost:8080/elec?open=configuration;path=cover\_elec:Cover". The page has a yellow header bar with the text "OntoManager @ Mercator Telescope". Below the header is a navigation bar with a blue circular icon and several tabs: "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Electronics" tab is currently selected. The main content area displays a hierarchical tree structure. The root node is "Cover", which is expanded to show three sub-nodes: "I/O modules", "terminals", and "connectors". Each sub-node has a small square icon with a plus sign next to it. Below the "Cover" node, there are four more nodes: "M1", "M3", "Telemetry", and "Timing", each with a similar plus icon. A mouse cursor is pointing at the "Cover" node.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

The screenshot displays the OntoManager @ Mercator Telescope web application. The browser address bar shows the URL: `localhost:8080/elec?open=configuration;path=cover_elec:Cover::I%2FO%20modules`. The application's main header is orange and contains the title "OntoManager @ Mercator Telescope". Below the header is a navigation bar with tabs: "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Electronics" tab is currently selected. The main content area shows a hierarchical tree structure under the "Cover" ontology. The tree is expanded to show the "I/O modules" category, which contains a list of slots from "slot0" to "slot13". Below the slots are "terminals" and "connectors". At the bottom of the tree, there are four additional categories: "M1", "M3", "Telemetry", and "Timing". A mouse cursor is pointing at the "I/O modules" category.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar displays the URL: `localhost:8080/elec?open=configuration;path=cover_elec:Cover::terminals`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes tabs for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The main content area displays a hierarchical tree structure under the "Cover" node. The tree includes the following components:

- IO modules**
  - slot0
  - slot1
  - slot2
  - slot3
  - slot4
  - slot5
  - slot6
  - slot7
  - slot8
  - slot9
  - slot10
  - slot11
  - slot12
  - slot13
- terminals**
  - PE
  - L
  - II
  - 24V
  - GND
- connectors**
  - M1
  - M3
  - Telemetry



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

The screenshot displays the OntoManager @ Mercator Telescope web application. The browser address bar shows the URL: `localhost:8080/elec?open=configuration;path=cover_elec:Cover::connectors`. The application has a navigation bar with tabs: Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, **Electronics**, and Software. The main content area shows a hierarchical tree view of the 'Cover' ontology. The tree is expanded to show the 'connectors' sub-ontology, which includes 'ECAT', 'T1', and 'T2'. Other visible sub-ontologies include 'I/O modules' (with slots 0-13) and 'terminals' (with PE, L, N, 24V, and GND).

OntoManager @ Mercator Telescope

Electronics

Cover

- I/O modules
  - slot0
  - slot1
  - slot2
  - slot3
  - slot4
  - slot5
  - slot6
  - slot7
  - slot8
  - slot9
  - slot10
  - slot11
  - slot12
  - slot13
- terminals
  - PE
  - L
  - N
  - 24V
  - GND
- connectors
  - ECAT
  - T1
  - T2

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot4

## OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

**Cover**

- I/O modules
  - slot0
  - slot1
  - slot2
  - slot3
  - slot4**
  - slot5
  - slot6
  - slot7
  - slot8
  - slot9
  - slot10
  - slot11
  - slot12
  - slot13
- terminals
  - PE
  - L
  - II
  - 24V
  - GND
- connectors
  - ECAT
  - T1
  - T2

### I/O Module instance slot4

SSI module for Top 1 and Top 2 encoders

#### Module type summary

ID	EL5002
Manufacturer	Beckhoff Automation
Description	2-channel SSI encoder
Used in	Cover (4)

Run LED1

D1+ D1- CL1+ CL1- D2+ D2- CL2+ CL2-

Power contact +24 V

Power contact 0 V

Data Clock

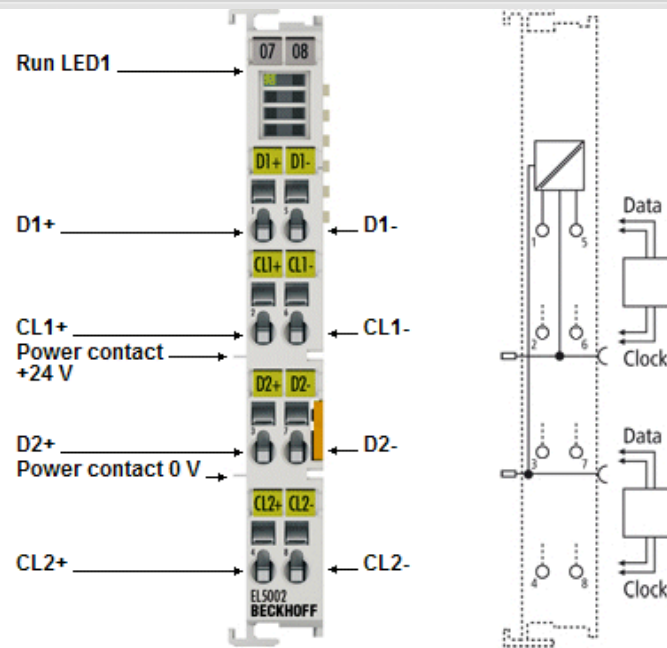
## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot4



**Connections**

Type (EL5002)				Instance		
Channel	Terminal	Symbol	Description	Symbol	Description	Connected to
1	1	D1+	Channel 1: Data +	TC:T1:SSID+	Top 1 SSI encoder Data +	Connector T1 : pin 6
	2	CL1+	Channel 1: Clock +	TC:T1:SSIC+	Top 1 SSI encoder Clock +	Connector T1 : pin 7
	5	D1-	Channel 1: Data -	TC:T1:SSID-	Top 1 SSI encoder Data -	Connector T1 : pin 14
	6	CL1-	Channel 1: Clock -	TC:T1:SSIC-	Top 1 SSI encoder Clock -	Connector T1 : pin 15
2	3	D2+	Channel 2: Data +	TC:T2:SSID+	Top 2 SSI encoder Data +	Connector T2 : pin 6
	4	CL2+	Channel 2: Clock +	TC:T2:SSIC+	Top 2 SSI encoder Clock +	Connector T2 : pin 7
	7	D2-	Channel 2: Data -	TC:T2:SSID-	Top 2 SSI encoder Data -	Connector T2 : pin 14
	8	CL2-	Channel 2: Clock -	TC:T2:SSIC-	Top 2 SSI encoder Clock -	Connector T2 : pin 15

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=ConnectorInstance;qname=cover\_elec:connectors.T1

## OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

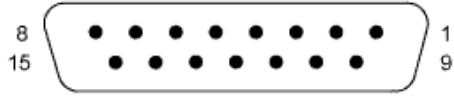
**Cover**

- I/O modules
  - + slot0
  - + slot1
  - + slot2
  - + slot3
  - + slot4
  - + slot5
  - + slot6
  - + slot7
  - + slot8
  - + slot9
  - + slot10
  - + slot11
  - + slot12
  - + slot13
- terminals
  - + PE
  - + L
  - + N
  - + 24V
  - + GND
- connectors
  - + ECAT
  - + T1
  - + T2

### Connector instance T1

Connector type summary

ID	D-sub 15 F
Gender	female
Manufacturer	ITT Corporation
Description	D-sub 15 female connector
Fits to	D-sub 15 M
Used in	Cover (8), M1 (1), M3 (2)



DA-15S (Female Socket Front View)

### Connections

Type (D-sub 15 F)			Instance		
Pin	Symbol	Description	Symbol	Description	Connected to
1	1	Pin 1	TC:T1:GND HM	Top 1 GND of holding magnet	Cover : terminal GND
2	2	Pin 2	TC:T1:GND MOT	Top 1 GND of motor	Cover : terminal GND
3	3	Pin 3	TC:T1:MMON	Top 1 motor monitor	
4	4	Pin 4	TC:T1:MDIR	Top 1 motor direction	I/O module slot1 : terminal 1
5	5	Pin 5	TC:T1:GND ENC	Top 1 GND of encoder	Cover : terminal GND



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=ConnectorInstance;qname=cover\_elec:connectors.T1

**slot5**  
**slot6**  
**slot7**  
**slot8**  
**slot9**  
**slot10**  
**slot11**  
**slot12**  
**slot13**

**terminals**  
**PE**  
**L**  
**N**  
**24V**  
**GND**

**connectors**  
**ECAT**  
**T1**  
**T2**  
**T3**  
**T4**  
**B1**  
**B2**  
**B3**  
**B4**  
**M1**  
**M3**  
**Telemetry**  
**Timing**

Description	D-sub 15 female connector
Fits to	D-sub 15 M
Used in	Cover (8), M1 (1), M3 (2)

8 15 1 9

DA-15S (Female Socket Front View)

**Connections**

Type (D-sub 15 F)			Instance		
Pin	Symbol	Description	Symbol	Description	Connected to
1	1	Pin 1	TC:T1:GND HM	Top 1 GND of holding magnet	Cover : terminal GND
2	2	Pin 2	TC:T1:GND MOT	Top 1 GND of motor	Cover : terminal GND
3	3	Pin 3	TC:T1:MMON	Top 1 motor monitor	
4	4	Pin 4	TC:T1:MDIR	Top 1 motor direction	I/O module slot1 : terminal 1
5	5	Pin 5	TC:T1:GND ENC	Top 1 GND of encoder	Cover : terminal GND
6	6	Pin 6	TC:T1:SSID+	Top 1 SSI Data +	I/O module slot4 : terminal 1
7	7	Pin 7	TC:T1:SSIC+	Top 1 SSI Clock +	I/O module slot4 : terminal 2
8	8	Pin 8	TC:T1:PE	Top 1 Earth	Cover : terminal PE
9	9	Pin 9	TC:T1:+24V HM	Top 1 +24V of holding magnet	I/O module slot12 : terminal 2
10	10	Pin 10	TC:T1:+24V MOT	Top 1 +24V of motor	I/O module slot8 : terminal 2
11	11	Pin 11	TC:T1:MSPEED	Top 1 motor speed	I/O module slot2 : terminal 1
12	12	Pin 12	TC:T1:+24V ENC	Top 1 +24V of encoder	Cover : terminal 24V
13	13	Pin 13	TC:T1:SSISTS	Top 1 SSI status	I/O module slot3 : terminal 1
14	14	Pin 14	TC:T1:SSID-	Top 1 SSI Data -	I/O module slot4 : terminal 5
15	15	Pin 15	TC:T1:SSIC-	Top 1 SSI Clock -	I/O module slot4 : terminal 6

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot3

## OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

**Cover**

- I/O modules
  - + slot0
  - + slot1
  - + slot2
  - + slot3
  - + slot4
  - + slot5
  - + slot6
  - + slot7
  - + slot8
  - + slot9
  - + slot10
  - + slot11
  - + slot12
  - + slot13
- terminals
  - + PE
  - + L
  - + N
  - + 24V
  - + GND
- connectors
  - + ECAT
  - + T1
  - + T2

### I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

#### System properties

**Satisfies** `cover_sys:panelDesign.requirements.absFeedbackStatus`

#### Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

The diagram illustrates the terminal block and internal circuitry of the I/O module. On the left, terminal labels point to specific pins: Signal LED1, Signal LED3, Signal LED5, Signal LED7, Input 1, Input 3, and Power contact +24 V. On the right, labels point to Signal LED2, Signal LED4, Signal LED6, Signal LED8, Input 2, and Input 4. The internal circuitry shows the connection of these inputs to the module's logic, including a 24V DC supply and negative switching components.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot3

**System properties**

Satisfies [cover\\_sys:panelDesign.requirements.absFeedbackStatus](#)

**Module type summary**

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

**terminals**

- + slot3
- + slot4
- + slot5
- + slot6
- + slot7
- + slot8
- + slot9
- + slot10
- + slot11
- + slot12
- + slot13

**connectors**

- + ECAT
- + T1
- + T2
- + T3
- + T4
- + B1
- + B2
- + B3
- + B4

**connections**

- + M1
- + M3
- + Telemetry

**Diagram:**

Signal LED1  
Signal LED3  
Signal LED5  
Signal LED7

Signal LED2  
Signal LED4  
Signal LED6  
Signal LED8

Input 1  
Input 2  
Input 3  
Power contact +24 V  
Input 4  
Input 5  
Power contact 0 V  
Input 6  
Input 7  
Input 8

EL1088  
BECKHOFF

**Connections**

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Systems design

The screenshot shows a web browser window with the URL `localhost:8080/sys?show=requirement;qname=cover_sys:panelDesign.requirements.absFeedbackStatus`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Systems" link is highlighted with a mouse cursor. On the left sidebar, a tree view shows the hierarchy: "Cover" (expanded), "M1", "M3", "Telemetry", and "Timing". The main content area displays the requirement "Requirement `absFeedbackStatus`" with a description: "The status of the absolute feedback shall be known". Below this, a "Properties" table lists the relationships for the requirement.

Properties	
Derives	
Derived from	<ul style="list-style-type: none"><li>← <code>cover_sys:concept.requirements.monitorable</code></li><li>← <code>cover_sys:panelDesign.requirements.absFeedback</code></li></ul>
Satisfied by	<ul style="list-style-type: none"><li>• <code>cover_sys:panelDesign.parts.encoder</code></li><li>• <code>cover_elec:slot3</code></li></ul>
Declared by	<ul style="list-style-type: none"><li>• <code>cover_sys:panelDesign</code></li></ul>



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Systems design

The screenshot shows a web browser window with the URL `localhost:8080/sys?show=requirement;qname=cover_sys:panelDesign.requirements.absFeedbackStatus`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, **Systems**, Mechanics, Electronics, and Software. On the left sidebar, there is a tree view with expandable items: Cover, M1, M3, Telemetry, and Timing. The main content area displays the "Requirement `absFeedbackStatus`". Below the title, a yellow box contains the text: "The status of the absolute feedback shall be known". Under the "Properties" section, there is a table with the following data:

Derives	
Derived from	<ul style="list-style-type: none"><li>← <code>cover_sys:concept.requirements.monitorable</code></li><li>← <code>cover_sys:panelDesign.requirements.absFeedback</code></li></ul>
Satisfied by	<ul style="list-style-type: none"><li>• <code>cover_sys:panelDesign.parts.encoder</code></li><li>• <code>cover_elec:slot3</code></li></ul>
Declared by	<ul style="list-style-type: none"><li>• <code>cover_sys:panelDesign</code></li></ul>

A mouse cursor is pointing at the `cover_sys:panelDesign` link in the "Declared by" row.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Systems design

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query **Systems** Mechanics Electronics Software

Design **panelDesign**

The design of the telescope cover panels

Requirements derivation matrix

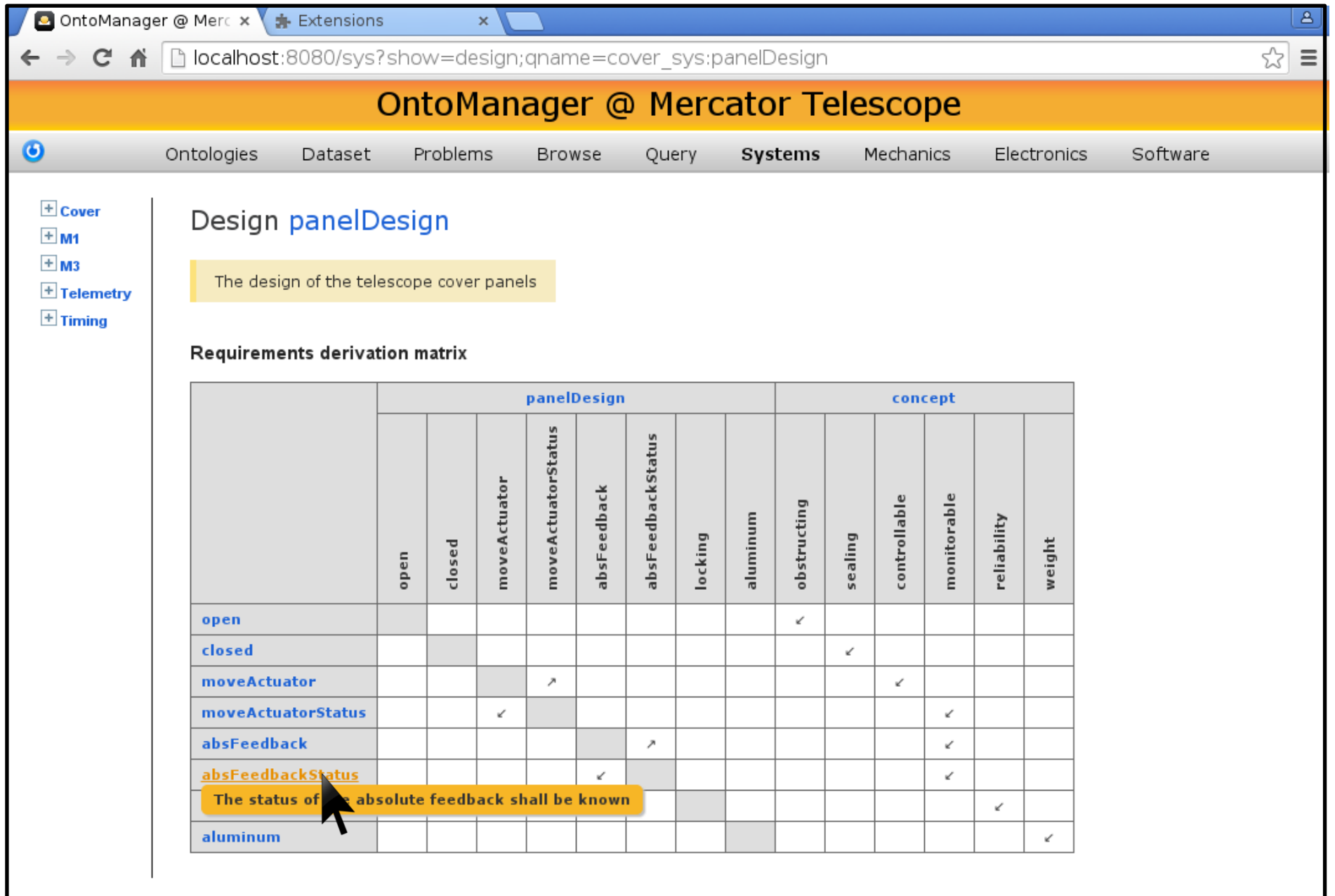
	panelDesign								concept					
	open	closed	moveActuator	moveActuatorStatus	absFeedback	absFeedbackStatus	locking	aluminum	obstructing	sealing	controllable	monitorable	reliability	weight
open									↙					
closed										↙				
moveActuator				↗							↙			
moveActuatorStatus			↙									↙		
absFeedback						↗							↙	
absFeedbackStatus					↙							↙		
													↙	
aluminum														↙

The status of the absolute feedback shall be known

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Systems design



OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query **Systems** Mechanics Electronics Software

**Design panelDesign**

The design of the telescope cover panels

**Requirements derivation matrix**

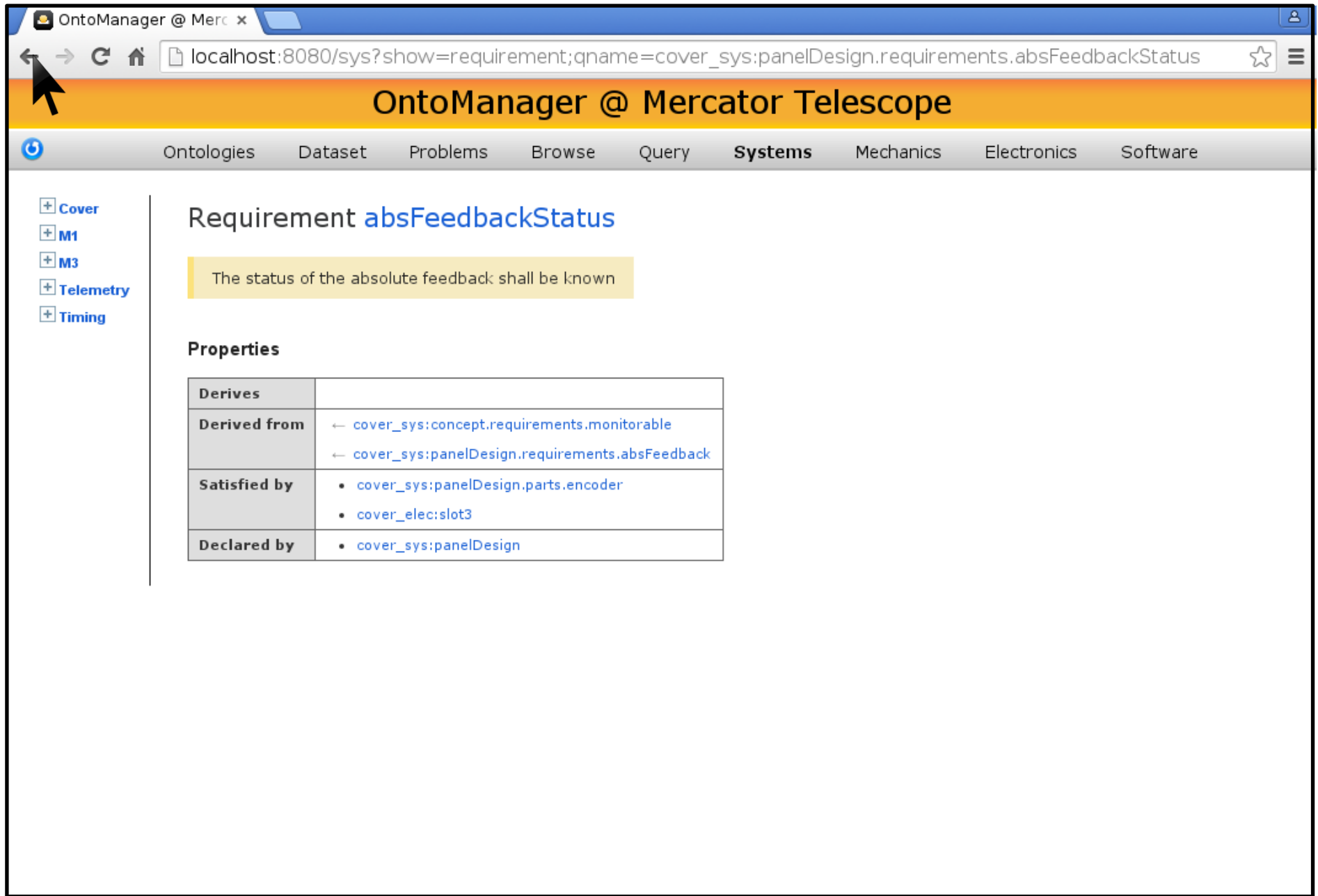
	panelDesign								concept					
	open	closed	moveActuator	moveActuatorStatus	absFeedback	absFeedbackStatus	locking	aluminum	obstructing	sealing	controllable	monitorable	reliability	weight
open									↙					
closed										↙				
moveActuator				↗							↙			
moveActuatorStatus			↙									↙		
absFeedback						↗							↙	
absFeedbackStatus						↙							↙	
													↙	
aluminum														↙

The status of the absolute feedback shall be known

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Systems design



The screenshot shows a web browser window with the title 'OntoManager @ Merc'. The address bar shows the URL 'localhost:8080/sys?show=requirement;qname=cover\_sys:panelDesign.requirements.absFeedbackStatus'. The page has a yellow header with the title 'OntoManager @ Mercator Telescope'. Below the header is a navigation bar with tabs: 'Ontologies', 'Dataset', 'Problems', 'Browse', 'Query', 'Systems', 'Mechanics', 'Electronics', and 'Software'. The 'Systems' tab is selected. On the left side, there is a sidebar with a tree view showing a hierarchy: 'Cover' (expanded), 'M1', 'M3', 'Telemetry', and 'Timing'. The main content area displays the 'Requirement absFeedbackStatus'. It includes a description: 'The status of the absolute feedback shall be known'. Below this, there is a section titled 'Properties' which contains a table with the following data:

Derives	
Derived from	<ul style="list-style-type: none"><li>← cover_sys:concept.requirements.monitorable</li><li>← cover_sys:panelDesign.requirements.absFeedback</li></ul>
Satisfied by	<ul style="list-style-type: none"><li>• cover_sys:panelDesign.parts.encoder</li><li>• cover_elec:slot3</li></ul>
Declared by	<ul style="list-style-type: none"><li>• cover_sys:panelDesign</li></ul>



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot3

## OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

**Cover**

- I/O modules
  - slot0
  - slot1
  - slot2
  - slot3
  - slot4
  - slot5
  - slot6
  - slot7
  - slot8
  - slot9
  - slot10
  - slot11
  - slot12
  - slot13
- terminals
  - PE
  - L
  - II
  - 24V
  - GND
- connectors
  - ECAT
  - T1
  - T2

### I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

#### System properties

Satisfies [cover\\_sys:panelDesign.requirements.absFeedbackStatus](#)

#### Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	<a href="#">Cover</a> (1), <a href="#">M3</a> (1)

The diagram illustrates the terminal block and internal circuitry of the I/O module. On the left, terminal labels point to specific pins: Signal LED1, Signal LED3, Signal LED5, Signal LED7, Input 1, Input 3, and Power contact +24 V. On the right, labels point to Signal LED2, Signal LED4, Signal LED6, Signal LED8, Input 2, and Input 4. The internal circuitry shows the connection of these inputs to the module's logic, including a 24V DC supply and negative switching components.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Mercator Telescope

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot3

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

### I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

#### System properties

Satisfies [cover\\_sys:panelDesign.requirements.absFeedbackStatus](#)

#### Module type summary

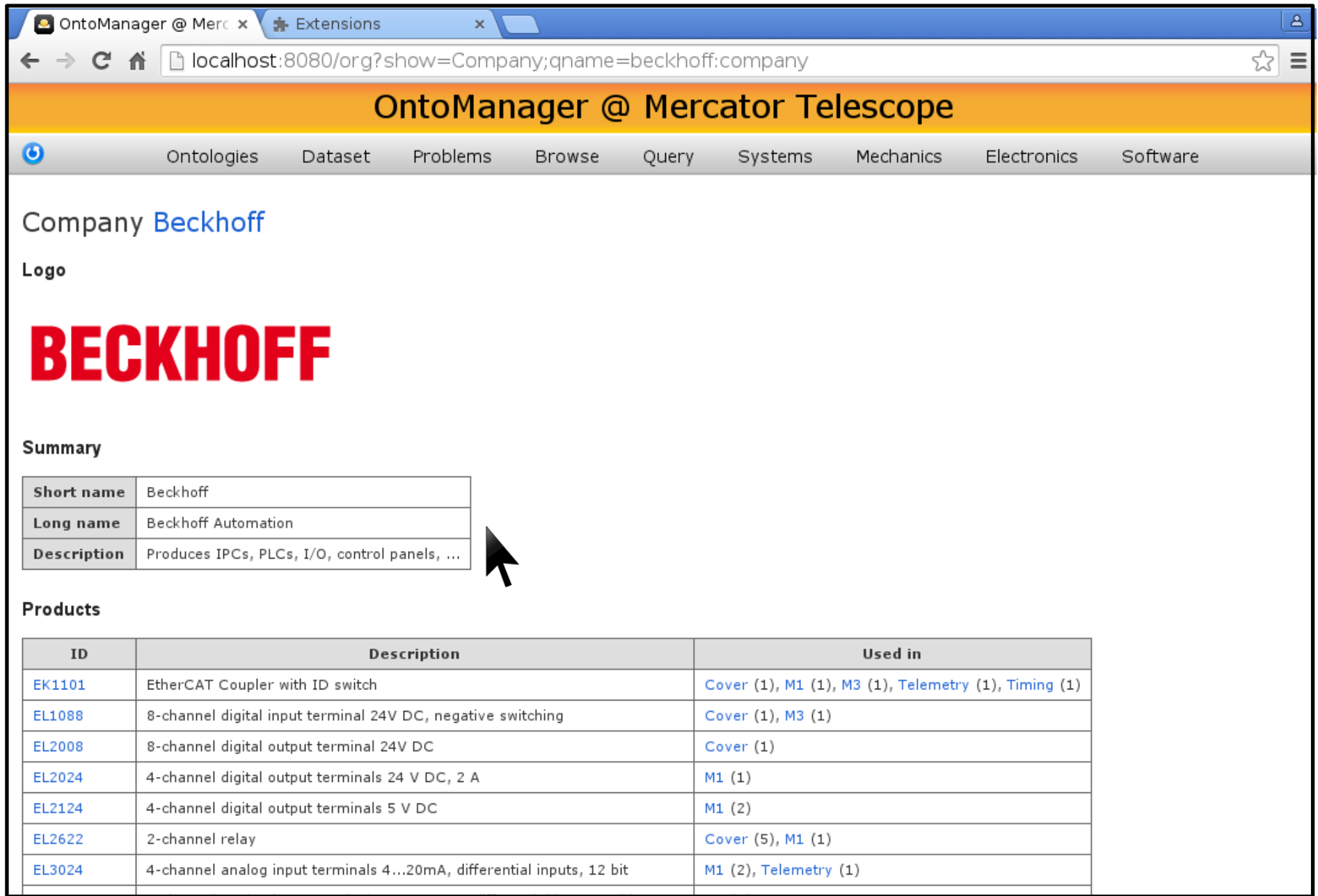
ID	EL1088
Manufacturer	<a href="#">Beckhoff Automation</a>
Description	8-channel digital input terminal 24V DC, negative switching
Used in	<a href="#">Cover</a> (1), <a href="#">M3</a> (1)

Signal LED1  
Signal LED3  
Signal LED5  
Signal LED7  
Signal LED2  
Signal LED4  
Signal LED6  
Signal LED8  
Input 1  
Input 2  
Input 3  
Power contact +24 V  
Input 4

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions


# Electrical design



The screenshot shows a web browser window with the URL `localhost:8080/org?show=Company;qname=beckhoff:company`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. The main content area displays the "Company Beckhoff" profile, including a logo and a summary table. A mouse cursor is pointing at the summary table. Below the summary table is a "Products" section with a table listing various products and their usage.

### Company Beckhoff

Logo



#### Summary

Short name	Beckhoff
Long name	Beckhoff Automation
Description	Produces IPCs, PLCs, I/O, control panels, ...

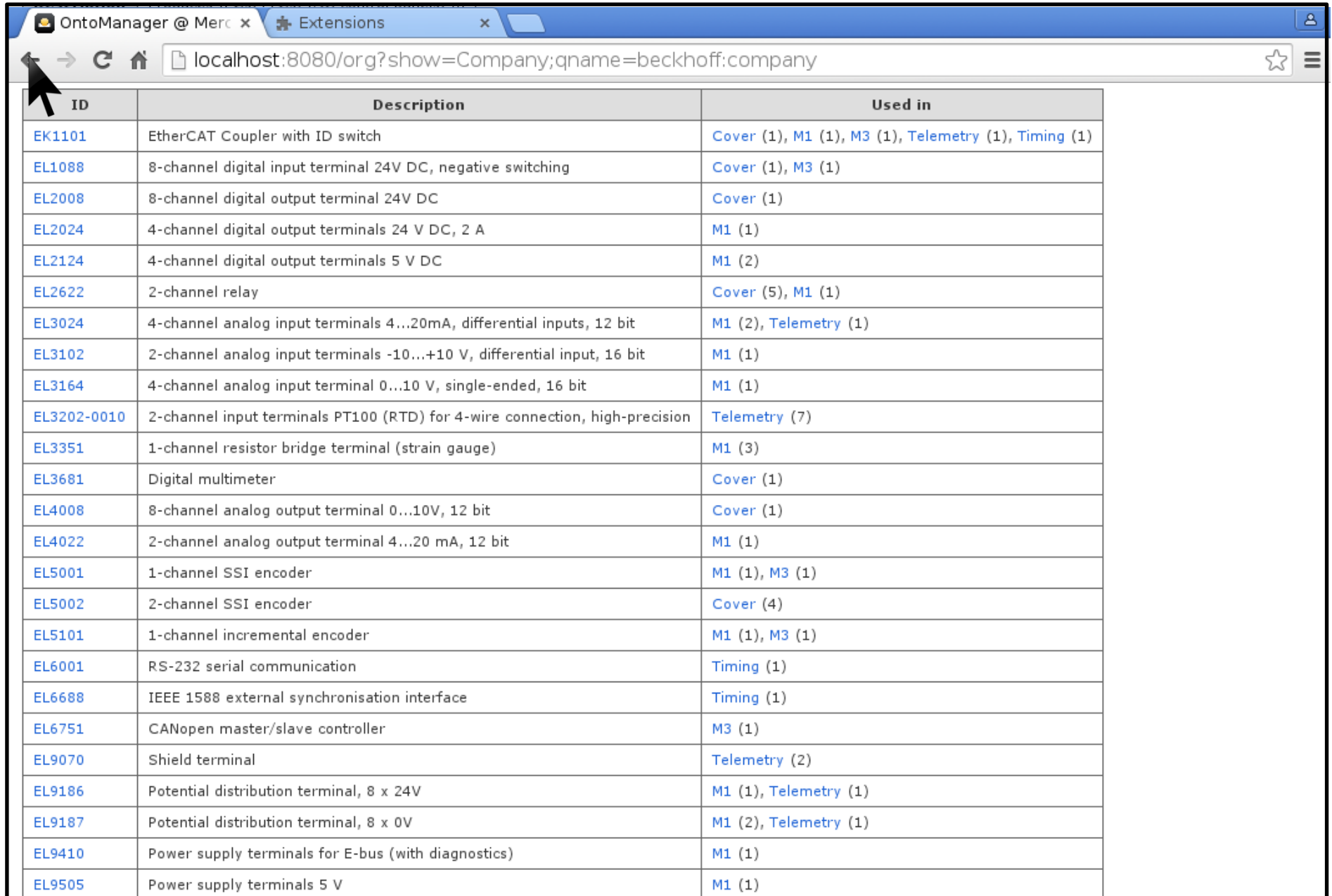
#### Products

ID	Description	Used in
<a href="#">EK1101</a>	EtherCAT Coupler with ID switch	<a href="#">Cover</a> (1), <a href="#">M1</a> (1), <a href="#">M3</a> (1), <a href="#">Telemetry</a> (1), <a href="#">Timing</a> (1)
<a href="#">EL1088</a>	8-channel digital input terminal 24V DC, negative switching	<a href="#">Cover</a> (1), <a href="#">M3</a> (1)
<a href="#">EL2008</a>	8-channel digital output terminal 24V DC	<a href="#">Cover</a> (1)
<a href="#">EL2024</a>	4-channel digital output terminals 24 V DC, 2 A	<a href="#">M1</a> (1)
<a href="#">EL2124</a>	4-channel digital output terminals 5 V DC	<a href="#">M1</a> (2)
<a href="#">EL2622</a>	2-channel relay	<a href="#">Cover</a> (5), <a href="#">M1</a> (1)
<a href="#">EL3024</a>	4-channel analog input terminals 4...20mA, differential inputs, 12 bit	<a href="#">M1</a> (2), <a href="#">Telemetry</a> (1)

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design



The screenshot shows a web browser window with the title "OntoManager @ Merc" and a tab for "Extensions". The address bar displays "localhost:8080/org?show=Company;qname=beckhoff:company". A mouse cursor is pointing at the first row of the table. The table has three columns: "ID", "Description", and "Used in". It lists various Beckhoff electrical components and their associated modules.

ID	Description	Used in
<a href="#">EK1101</a>	EtherCAT Coupler with ID switch	<a href="#">Cover</a> (1), <a href="#">M1</a> (1), <a href="#">M3</a> (1), <a href="#">Telemetry</a> (1), <a href="#">Timing</a> (1)
<a href="#">EL1088</a>	8-channel digital input terminal 24V DC, negative switching	<a href="#">Cover</a> (1), <a href="#">M3</a> (1)
<a href="#">EL2008</a>	8-channel digital output terminal 24V DC	<a href="#">Cover</a> (1)
<a href="#">EL2024</a>	4-channel digital output terminals 24 V DC, 2 A	<a href="#">M1</a> (1)
<a href="#">EL2124</a>	4-channel digital output terminals 5 V DC	<a href="#">M1</a> (2)
<a href="#">EL2622</a>	2-channel relay	<a href="#">Cover</a> (5), <a href="#">M1</a> (1)
<a href="#">EL3024</a>	4-channel analog input terminals 4...20mA, differential inputs, 12 bit	<a href="#">M1</a> (2), <a href="#">Telemetry</a> (1)
<a href="#">EL3102</a>	2-channel analog input terminals -10...+10 V, differential input, 16 bit	<a href="#">M1</a> (1)
<a href="#">EL3164</a>	4-channel analog input terminal 0...10 V, single-ended, 16 bit	<a href="#">M1</a> (1)
<a href="#">EL3202-0010</a>	2-channel input terminals PT100 (RTD) for 4-wire connection, high-precision	<a href="#">Telemetry</a> (7)
<a href="#">EL3351</a>	1-channel resistor bridge terminal (strain gauge)	<a href="#">M1</a> (3)
<a href="#">EL3681</a>	Digital multimeter	<a href="#">Cover</a> (1)
<a href="#">EL4008</a>	8-channel analog output terminal 0...10V, 12 bit	<a href="#">Cover</a> (1)
<a href="#">EL4022</a>	2-channel analog output terminal 4...20 mA, 12 bit	<a href="#">M1</a> (1)
<a href="#">EL5001</a>	1-channel SSI encoder	<a href="#">M1</a> (1), <a href="#">M3</a> (1)
<a href="#">EL5002</a>	2-channel SSI encoder	<a href="#">Cover</a> (4)
<a href="#">EL5101</a>	1-channel incremental encoder	<a href="#">M1</a> (1), <a href="#">M3</a> (1)
<a href="#">EL6001</a>	RS-232 serial communication	<a href="#">Timing</a> (1)
<a href="#">EL6688</a>	IEEE 1588 external synchronisation interface	<a href="#">Timing</a> (1)
<a href="#">EL6751</a>	CANopen master/slave controller	<a href="#">M3</a> (1)
<a href="#">EL9070</a>	Shield terminal	<a href="#">Telemetry</a> (2)
<a href="#">EL9186</a>	Potential distribution terminal, 8 x 24V	<a href="#">M1</a> (1), <a href="#">Telemetry</a> (1)
<a href="#">EL9187</a>	Potential distribution terminal, 8 x 0V	<a href="#">M1</a> (2), <a href="#">Telemetry</a> (1)
<a href="#">EL9410</a>	Power supply terminals for E-bus (with diagnostics)	<a href="#">M1</a> (1)
<a href="#">EL9505</a>	Power supply terminals 5 V	<a href="#">M1</a> (1)



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot3

## OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics **Electronics** Software

**Cover**

- I/O modules
  - slot0
  - slot1
  - slot2
  - slot3
  - slot4
  - slot5
  - slot6
  - slot7
  - slot8
  - slot9
  - slot10
  - slot11
  - slot12
  - slot13
- terminals
  - PE
  - L
  - N
  - 24V
  - GND
- connectors
  - ECAT
  - T1
  - T2

### I/O Module instance slot3

Digital input terminal to read the status of the SSI encoders of all 8 cover panels

#### System properties

**Satisfies** `cover_sys:panelDesign.requirements.absFeedbackStatus`

#### Module type summary

ID	EL1088
Manufacturer	Beckhoff Automation
Description	8-channel digital input terminal 24V DC, negative switching
Used in	Cover (1), M3 (1)

The diagram illustrates the physical and electrical layout of the I/O module. On the left, a vertical terminal block is shown with labels for various inputs and outputs: Signal LED1 through LED7 on the left, Signal LED2 through LED8 on the right, Input 1 and Input 2 in the middle, Input 3 and Input 4 at the bottom, and a Power contact +24 V at the very bottom. The terminal block is divided into sections labeled 07-08, I1-I2, I3-I4, and I5-I6. To the right of the terminal block, a schematic diagram shows the internal wiring, including signal lines, power lines, and ground connections, with components like resistors and diodes.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Electrical design

OntoManager @ Merc x

localhost:8080/elec?show=IoModuleInstance;qname=cover\_elec:slot3

**Connections**

Type (EL1088)				Instance		
Channel	Terminal	Symbol	Description	Symbol	Description	Connected to
1	1	I1	Input 1	TC:T1:SSISTS	Top 1 SSI encoder status	Connector T1 : pin 13
2	2	I2	Input 2	TC:T2:SSISTS	Top 2 SSI encoder status	Connector T2 : pin 13
3	3	I3	Input 3	TC:T3:SSISTS	Top 3 SSI encoder status	Connector T3 : pin 13
4	4	I4	Input 4	TC:T4:SSISTS	Top 4 SSI encoder status	Connector T4 : pin 13
5	5	I5	Input 5	TC:B1:SSISTS	Bottom SSI encoder status	Connector B1 : pin 13
6	6	I6	Input 6	TC:B2:SSISTS	Bottom SSI encoder status	Connector B2 : pin 13
7	7	I7	Input 7	TC:B3:SSISTS	Bottom SSI encoder status	Connector B3 : pin 13
8	8	I8	Input 8	TC:B4:SSISTS	Bottom SSI encoder status	Connector B4 : pin 13

**Interface**

Variable	Type	Description	Linked variable
input1	BOOL	Input 1	interface.parts.cover.parts.top.parts.p1.encoderErrorSignal
input2	BOOL	Input 2	interface.parts.cover.parts.top.parts.p2.encoderErrorSignal
input3	BOOL	Input 3	interface.parts.cover.parts.top.parts.p3.encoderErrorSignal
input4	BOOL	Input 4	interface.parts.cover.parts.top.parts.p4.encoderErrorSignal
input5	BOOL	Input 5	interface.parts.cover.parts.bottom.parts.p1.encoderErrorSignal
input6	BOOL	Input 6	interface.parts.cover.parts.bottom.parts.p2.encoderErrorSignal
input7	BOOL	Input 7	interface.parts.cover.parts.bottom.parts.p3.encoderErrorSignal
input8	BOOL	Input 8	interface.parts.cover.parts.bottom.parts.p4.encoderErrorSignal
WcState	BOOL	EtherCAT Working counter state	interface.parts.cover.parts.io.parts.slot3.wcState
InfoDataState	UINT	EtherCAT state (INIT, PREOP, OP, ...)	interface.parts.cover.parts.io.parts.slot3.infoData

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar displays `localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. On the left, a tree view shows the project structure: Tc2\_MC2, mtcs\_common, mtcs\_cover, mtcs\_m1, mtcs\_m3, mtcs\_telemetry, mtcs\_timing, and mtcs. The main content area displays the "FunctionBlock SM\_CoverPanel" with its internal structure and connections. A "Jump to:" sidebar on the right offers links to Variables, Methods, Implementation, and PLCopen XML serialization. Below the function block diagram, a "Variables" table lists the data points for the cover panel.

### FunctionBlock SM\_CoverPanel

Diagram showing the internal structure and connections of the SM\_CoverPanel function block:

- encoderErrorSignal (input) connected to actualStatus (output)
- initializationStatus (input) connected to statuses (output)
- operatorStatus (input) connected to parts (output)
- operatingStatus (input) connected to processes (output)
- config (input) connected to startOpening() (output)
- coverConfig (input) connected to startClosing() (output)

### Variables

Variable	Name	Type	Initial value	Address	Description	Qualif
VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

OntoManager @ Merc x

localhost:8080/soft?show=fb;qname=cover\_soft:mtcs\_cover.SM\_CoverPanel

**FunctionBlock SM\_CoverPanel**

**Variables**

Variable	Name	Type	Initial value	Address	Description	Qualif
VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC
	statuses	CoverPanelStatuses			Statuses of the state machine	
	parts	CoverPanelParts			Parts of the state machine	
	processes	CoverPanelProcesses			Processes of the state machine	

Jump to:

- Variables
- Methods
- Implementation
- PLCopen XML serialization



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

OntoManager @ Merc x

localhost:8080/soft?show=fb;qname=cover\_soft:mtcs\_cover.SM\_CoverPanel

VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC
	statuses	CoverPanelStatuses			Statuses of the state machine	
	parts	CoverPanelParts			Parts of the state machine	
	processes	CoverPanelProcesses			Processes of the state machine	

### Methods

- **startOpening()**

Comment	Start opening the panel						
Return type	RequestResults						
Interface	Variable	Name	Type	Initial value	Address	Description	Qualifiers
Implementation	startOpening := THIS^.processes.startOpening.request();						

- **startClosing()**

Comment	Start closing the panel						
Return type	RequestResults						
Interface	Variable	Name	Type	Initial value	Address	Description	Qualifiers
Implementation	startClosing := THIS^.processes.startClosing.request();						

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# Software design

OntoManager @ Merc x

localhost:8080/soft?show=fb;qname=cover\_soft:mtcs\_cover.SM\_CoverPanel

Implementation startOpening := THIS^.processes.startOpening.request();

- startClosing()

Comment	Start closing the panel						
Return type	RequestResults						
Interface	Variable	Name	Type	Initial value	Address	Description	Qualifiers
Implementation	startClosing := THIS^.processes.startClosing.request();						

Implementation

```
parts.axis(  
  isEnabled := operatorStatus.tech AND (operatingStatus.manual AND initializationStatus.initialized),  
  standstillTolerance := config.standstillTolerance);  
parts.motorRelay( isEnabled := parts.axis.isEnabled );  
statuses.busyStatus( isBusy := parts.axis.statuses.busyStatus.busy OR parts.motorRelay.statuses.busyStatus.busy );  
statuses.apertureStatus(  
  isOpen := (ABS(config.openPosition - parts.axis.actPos.degrees.value)) < config.openTolerance,  
  isClosed := (ABS(config.closedPosition - parts.axis.actPos.degrees.value)) < config.closedTolerance);  
statuses.healthStatus(  
  isGood := parts.axis.statuses.healthStatus.isGood AND (NOT(encoderErrorSignal)),  
  hasWarning := parts.axis.statuses.healthStatus.hasWarning);  
statuses.openingStatus(  
  isOpening := parts.axis.statuses.motionStatus.backward,  
  isClosing := parts.axis.statuses.motionStatus.forward);  
processes.startOpening( isEnabled := operatorStatus.tech AND (operatingStatus.manual AND initializationStatus.initialized) );  
processes.startClosing( isEnabled := operatorStatus.tech AND (operatingStatus.manual AND initializationStatus.initialized) );
```

PLCopen XML serialization

```
1 <pou name="SM_CoverPanel" pouType="functionBlock">  
2   <interface>  
3     <inputVars>  
4       <variable name="encoderErrorSignal" address="%I*">  
5         <type><BOOL /></type>  
6         <addData>
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

The screenshot shows the OntoManager @ Mercator Telescope web application. The browser address bar displays `localhost:8080/soft?show=fb;qname=cover_soft:mtcs_cover.SM_CoverPanel`. The application has a navigation bar with tabs: Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. On the left, a tree view shows the project structure with `mtcs_cover` selected. The main content area displays the `FunctionBlock SM_CoverPanel` with its inputs, outputs, and methods. A 'Jump to:' sidebar on the right offers links to Variables, Methods, Implementation, and PLCopen XML serialization. Below the function block, a 'Variables' table lists the data points used in the block.

### FunctionBlock SM\_CoverPanel

Input/Output	Variable Name	Type
Input	encoderErrorSignal	actualStatus
Input	initializationStatus	statuses
Input	operatorStatus	parts
Input	operatingStatus	processes
Input	config	
Input	coverConfig	
Output	startOpening()	
Output	startClosing()	

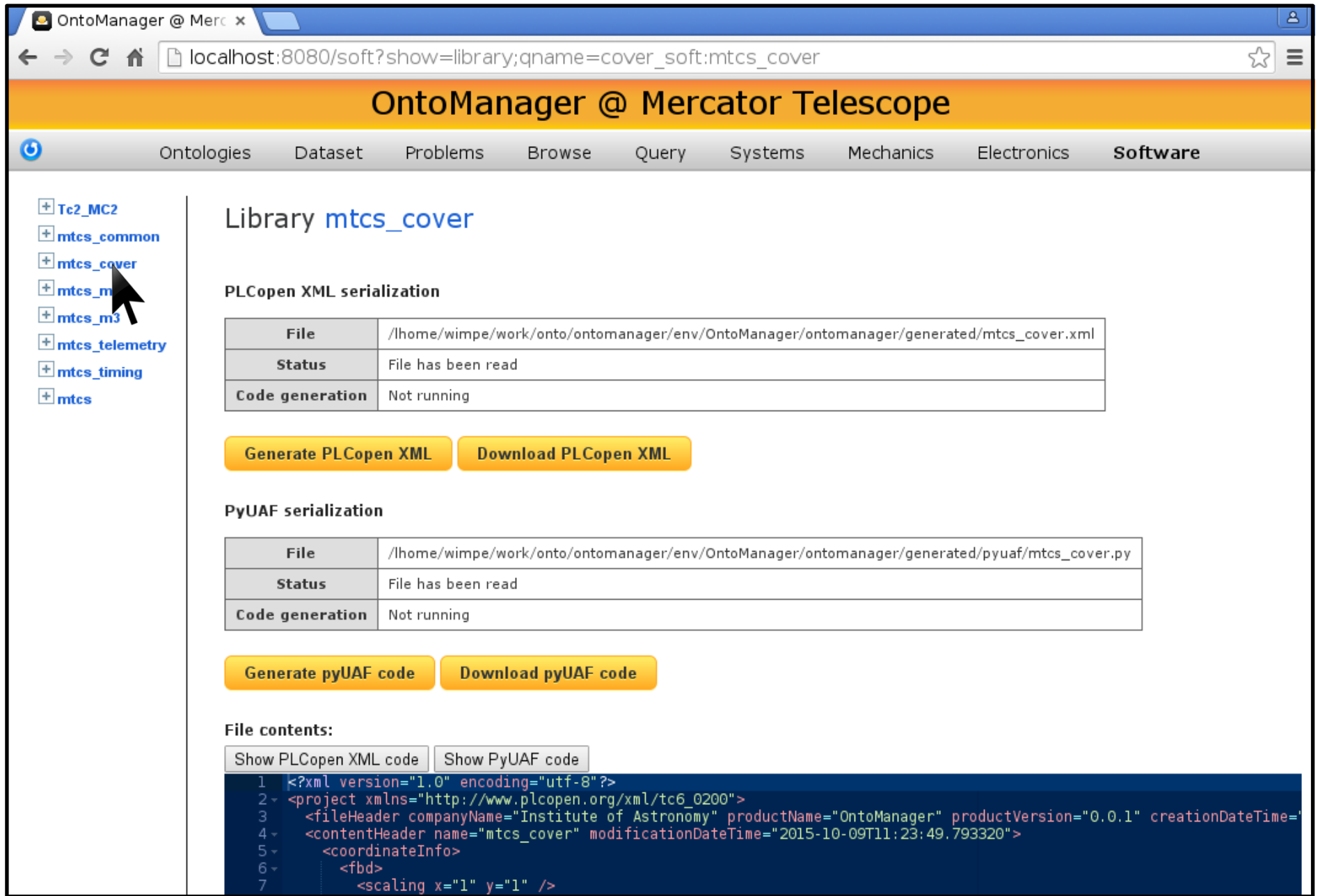
### Variables

Variable	Name	Type	Initial value	Address	Description	Qualif
VAR_INPUT	encoderErrorSignal	BOOL		%I*	Externally read error signal	OPC.UA.DA=1, OPC
VAR_IN_OUT	initializationStatus	InitializationStatus			INITIALIZED or INITIALIZING or ...	
	operatorStatus	OperatorStatus			TECH or OBSERVER or ...	
	operatingStatus	OperatingStatus			MANUAL or AUTO or NONE	
	config	CoverPanelConfig			Configuration of the panel	
	coverConfig	CoverConfig			Configuration of the cover	
VAR_OUTPUT	actualStatus	STRING			Current status description	OPC.UA.DA=1, OPC

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design



The screenshot shows a web browser window with the title "OntoManager @ Merc x". The address bar shows the URL "localhost:8080/soft?show=library;qname=cover\_soft:mtcs\_cover". The page has a yellow header with the text "OntoManager @ Mercator Telescope". Below the header is a navigation bar with tabs: "Ontologies", "Dataset", "Problems", "Browse", "Query", "Systems", "Mechanics", "Electronics", and "Software". The "Software" tab is selected. On the left side, there is a tree view of libraries. The "mtcs\_cover" library is selected, and a mouse cursor is pointing at it. The main content area shows the "Library mtcs\_cover" page. It has a section for "PLCopen XML serialization" with a table showing file information and buttons to generate or download the XML. Below this is a section for "PyUAF serialization" with a similar table and buttons. At the bottom, there is a "File contents:" section with two tabs: "Show PLCopen XML code" and "Show PyUAF code". The "Show PLCopen XML code" tab is active, displaying XML code for the mtcs\_cover library.

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics Electronics **Software**

Library **mtcs\_cover**

PLCopen XML serialization

File	/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml
Status	File has been read
Code generation	Not running

[Generate PLCopen XML](#) [Download PLCopen XML](#)

PyUAF serialization

File	/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py
Status	File has been read
Code generation	Not running

[Generate pyUAF code](#) [Download pyUAF code](#)

File contents:

Show PLCopen XML code Show PyUAF code

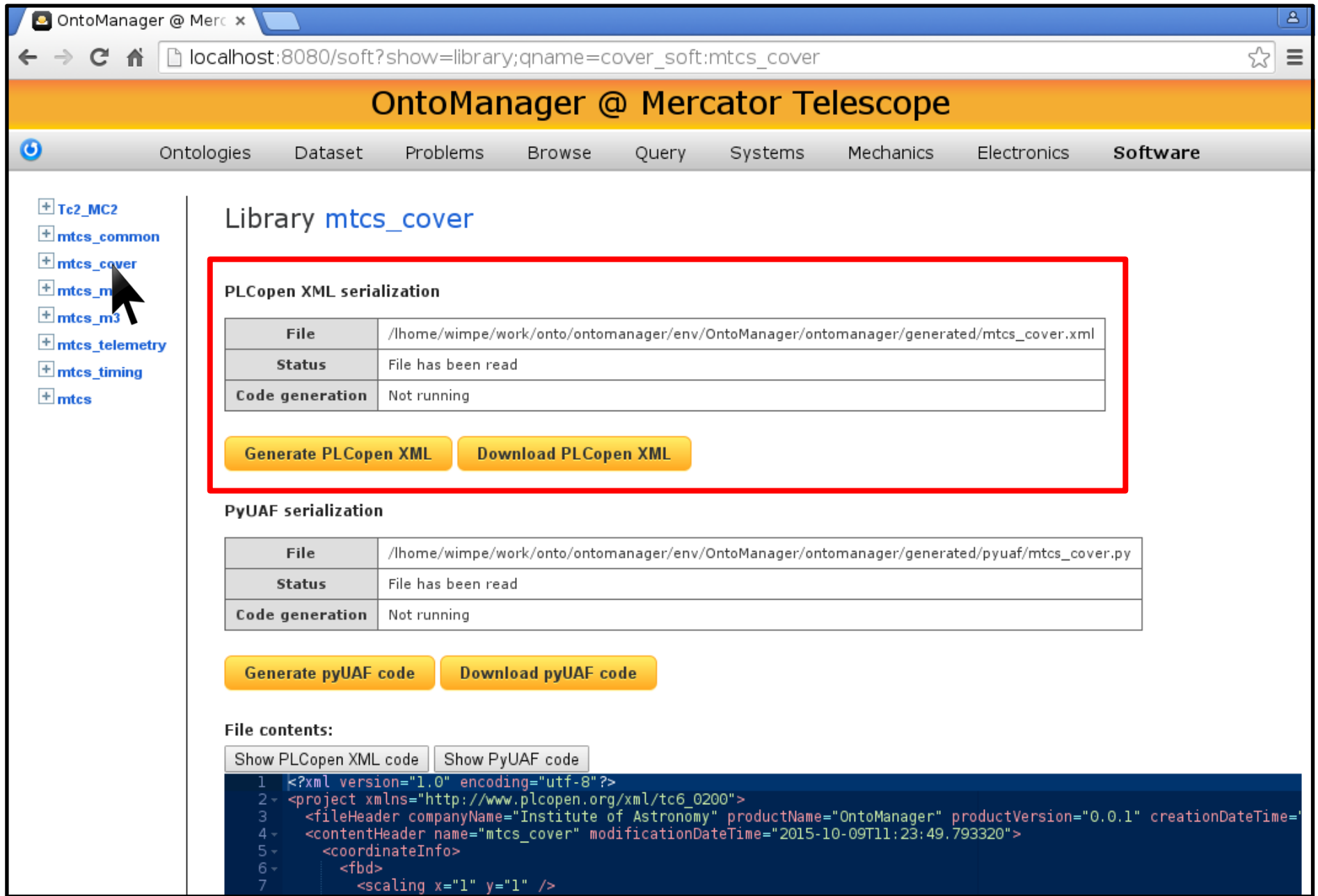
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <project xmlns="http://www.plcopen.org/xml/tc6_0200">
3   <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
4   <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
5     <coordinateInfo>
6       <fbid>
7       <scaling x="1" y="1" />
```



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design



The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar shows the URL `localhost:8080/soft?show=library;qname=cover_soft:mtcs_cover`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. The left sidebar shows a tree of ontologies, with `mtcs_cover` selected. The main content area displays the "Library mtcs\_cover" page. A red box highlights the "PLCopen XML serialization" section, which includes a table with the following data:

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml	File has been read	Not running

Below the table are two buttons: "Generate PLCopen XML" and "Download PLCopen XML".

The "PyUAF serialization" section is also visible, with a table showing the file `/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py` and its status "File has been read". Below this table are buttons for "Generate pyUAF code" and "Download pyUAF code".

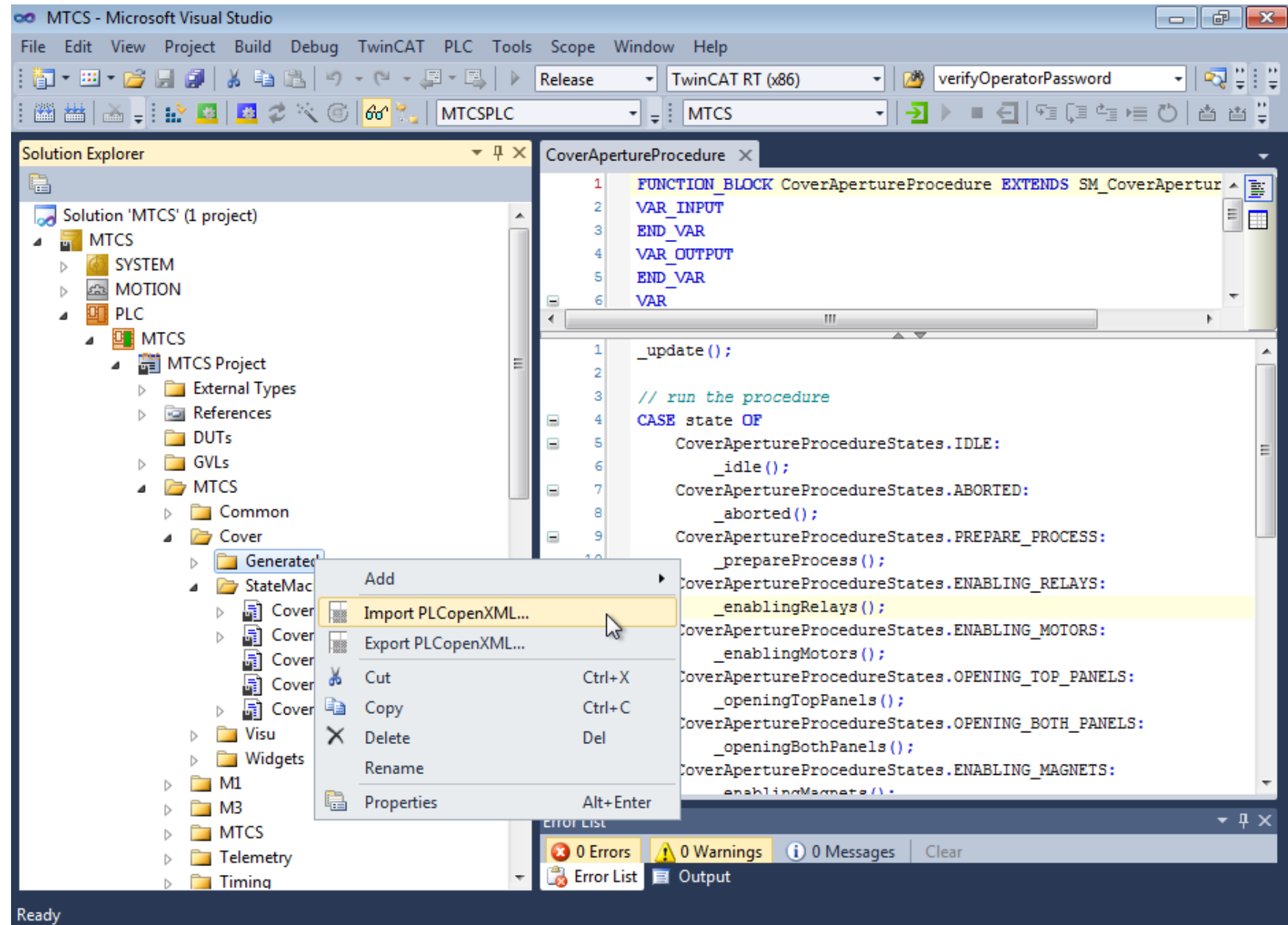
The "File contents:" section shows two tabs: "Show PLCopen XML code" and "Show PyUAF code". The "Show PLCopen XML code" tab is active, displaying the following XML code:

```
<?xml version="1.0" encoding="utf-8"?>
<project xmlns="http://www.plcopen.org/xml/tc6_0200">
  <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
  <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
    <coordinateInfo>
      <fbid>
        <scaling x="1" y="1" />
      </fbid>
    </coordinateInfo>
  </contentHeader>
</project>
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

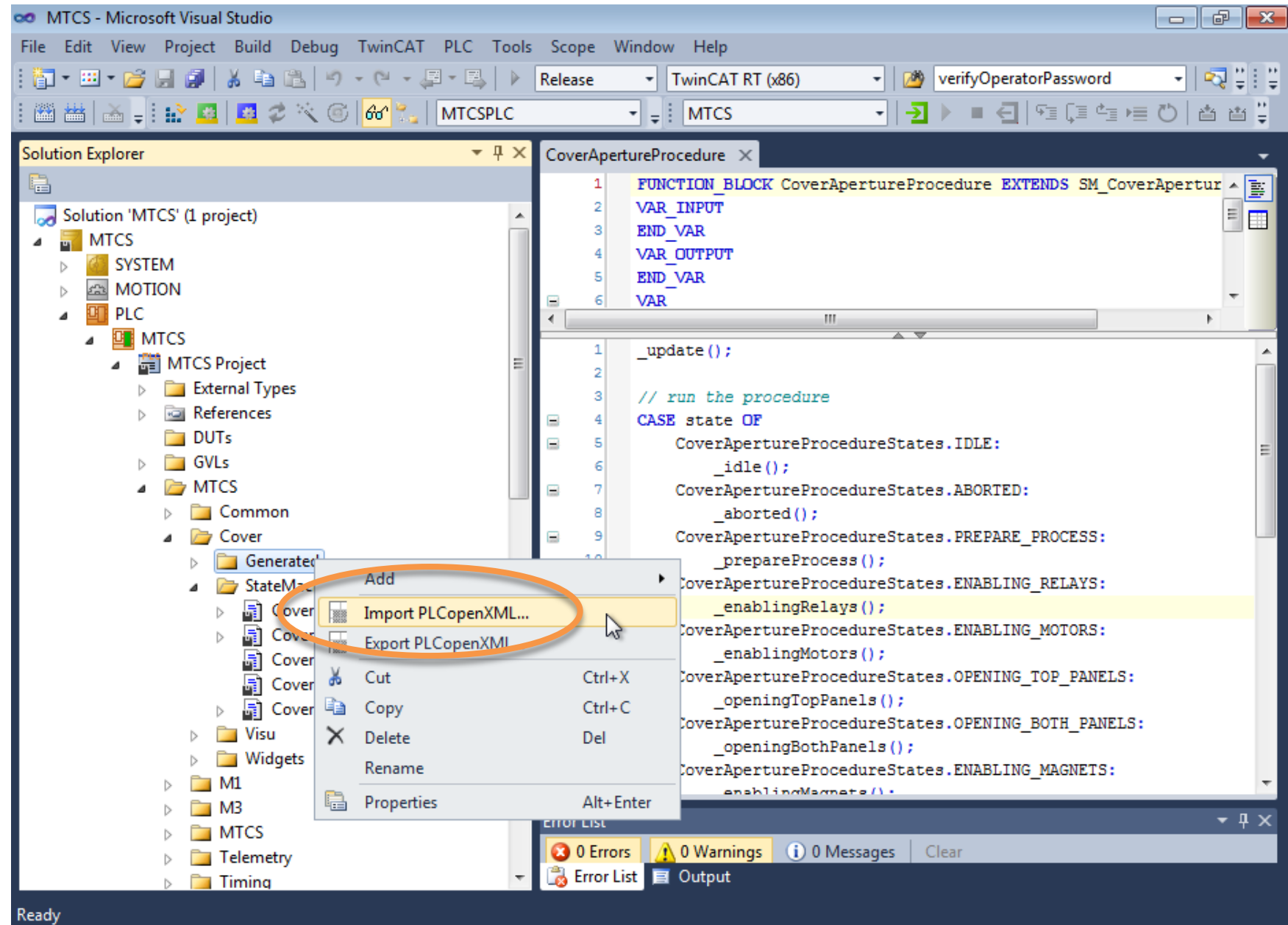
# Software design



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

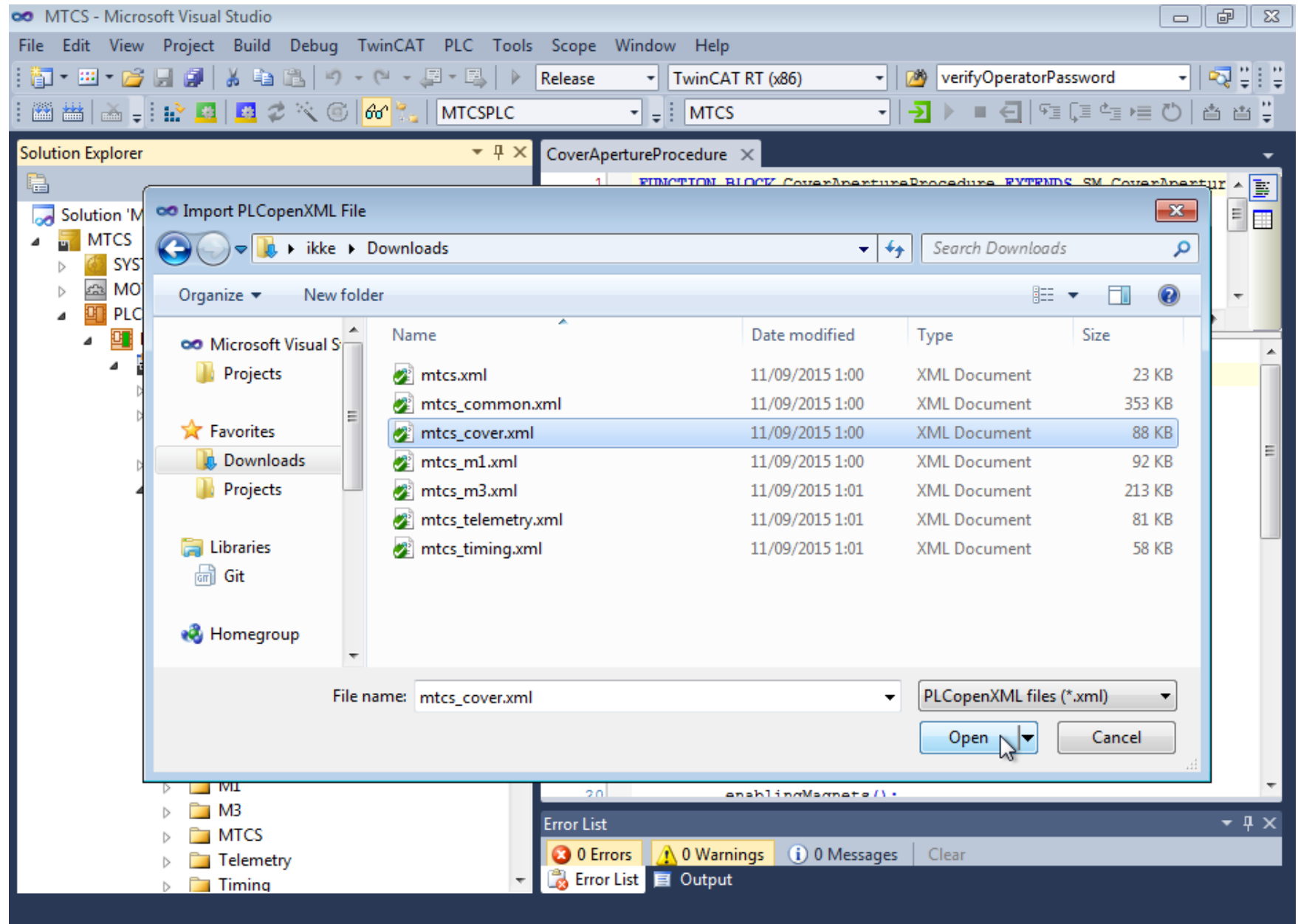
# Software design



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

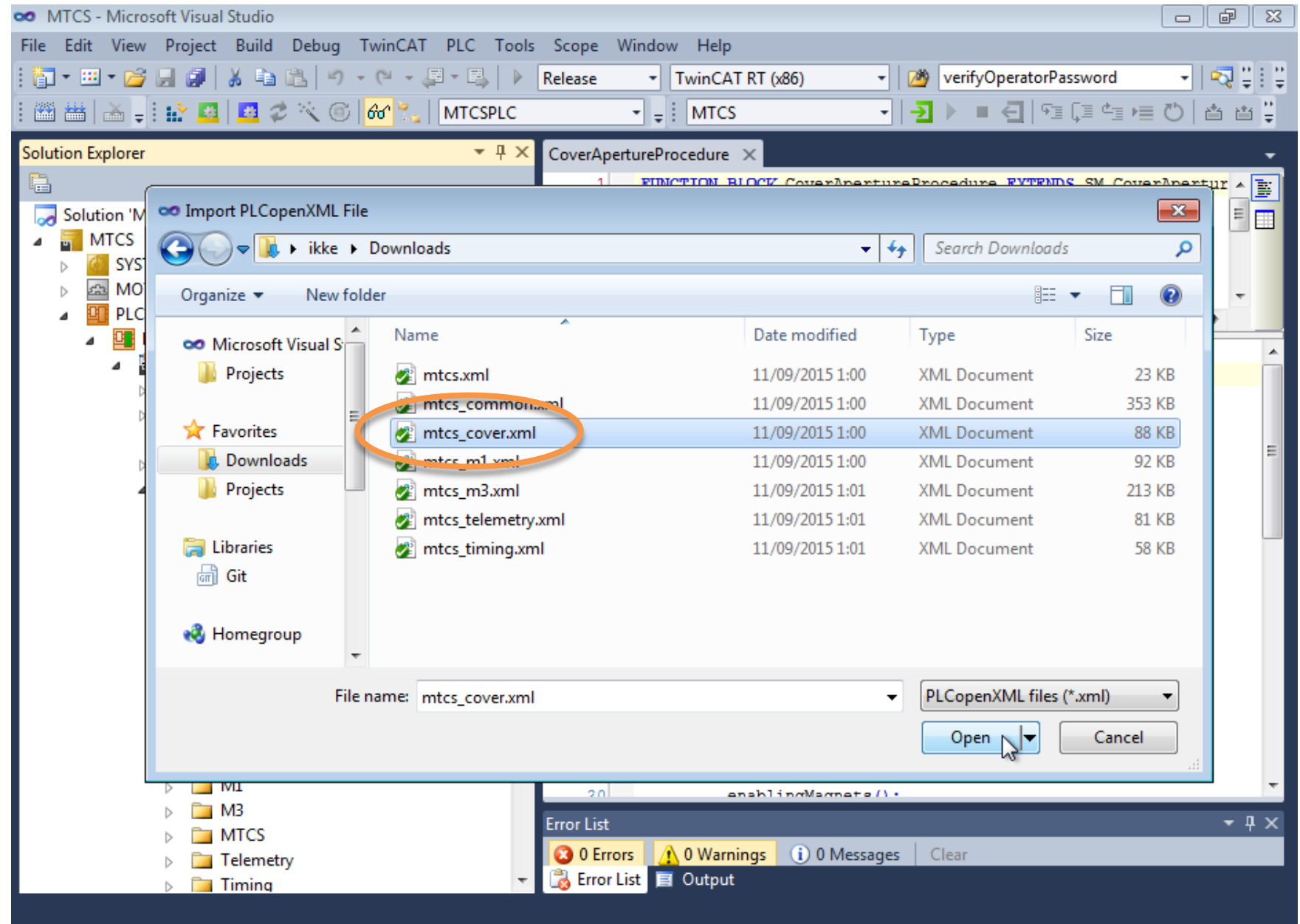




## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

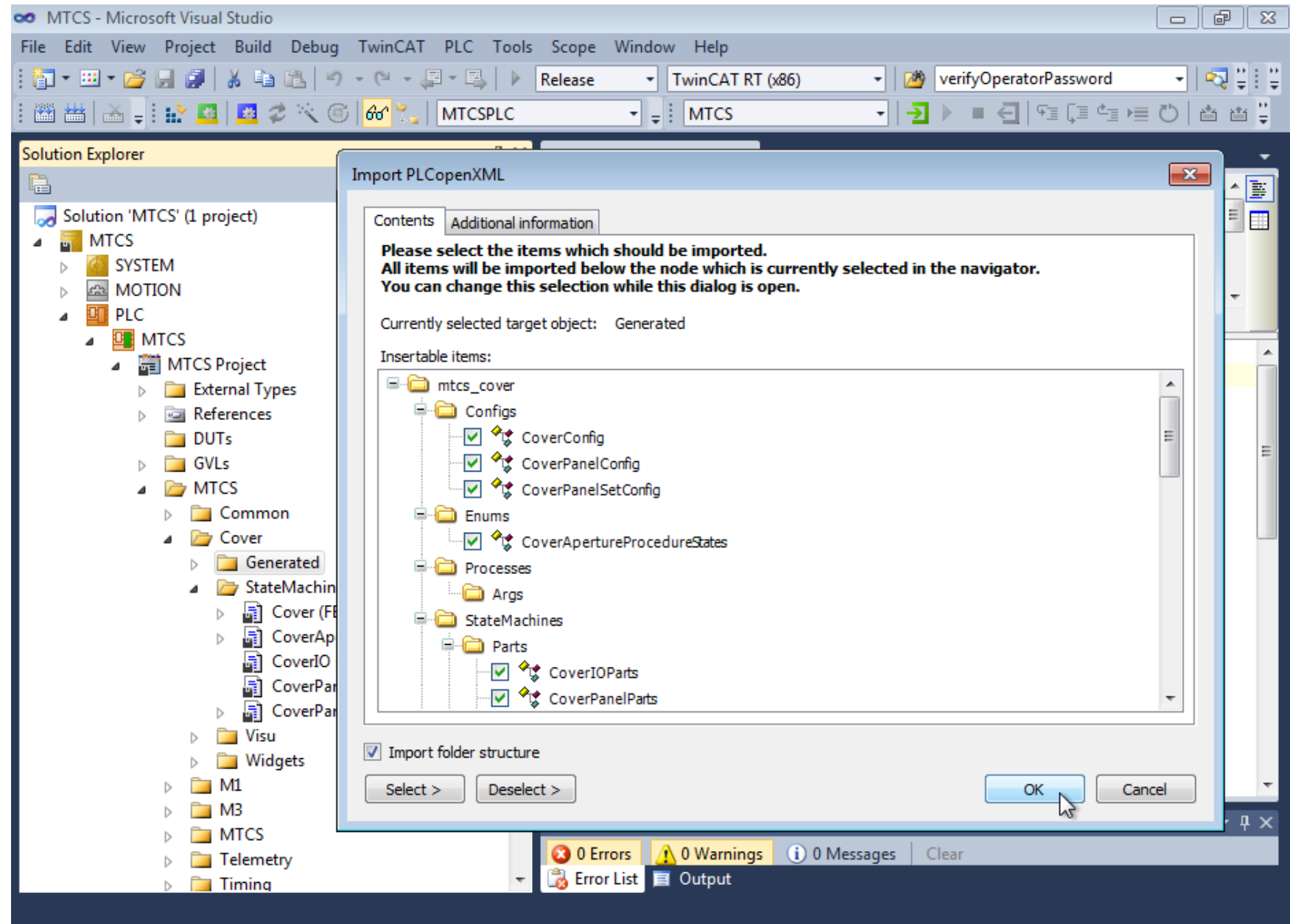
# Software design

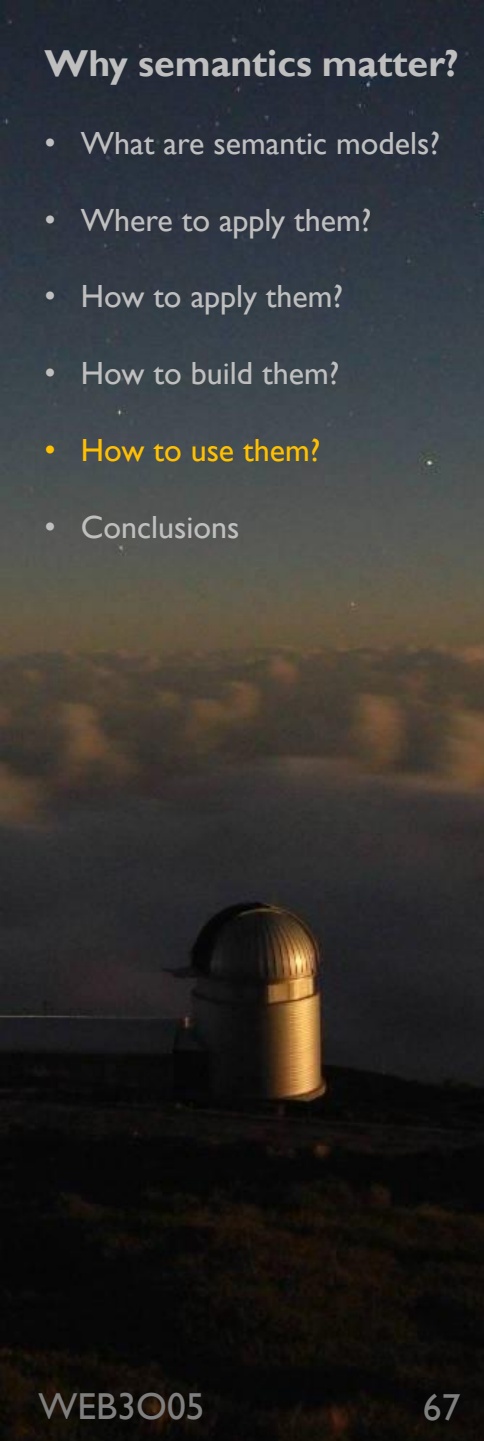


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

# Software design



A photograph of a telescope dome on a hill at sunset. The dome is illuminated by the low sun, creating a warm glow. The sky is filled with soft, orange and yellow clouds. The foreground is dark and silhouetted.

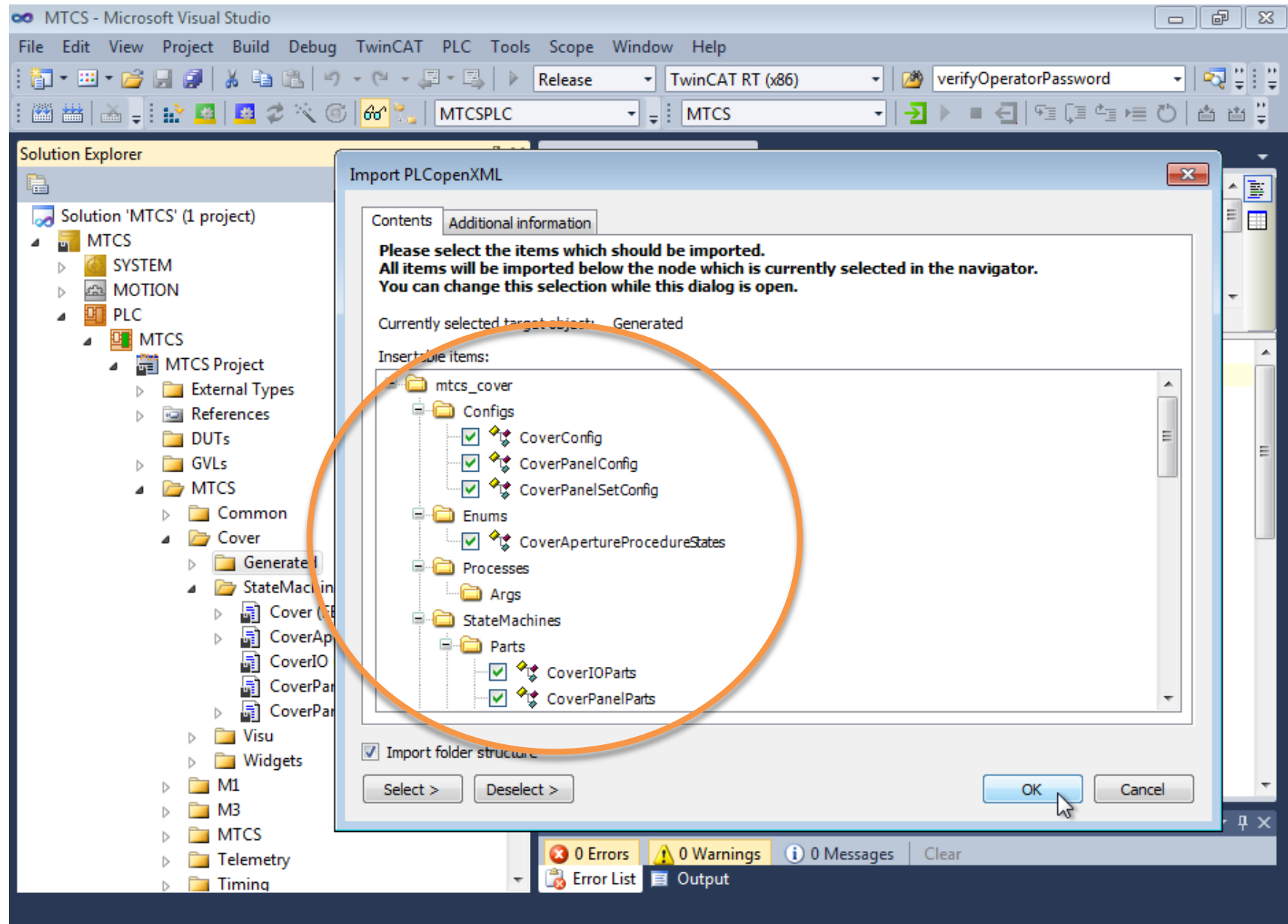
# Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

WEB3O05 67

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

The screenshot shows the OntoManager @ Mercator Telescope web application. The browser address bar shows the URL `localhost:8080/soft?show=library;qname=cover_soft:mtcs_cover`. The application has a navigation bar with tabs: Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. On the left, a sidebar lists ontologies: Tc2\_MC2, mtcs\_common, mtcs\_cover (highlighted by a mouse cursor), mtcs\_m, mtcs\_m3, mtcs\_telemetry, mtcs\_timing, and mtcs. The main content area displays the 'Library mtcs\_cover' page. It includes a table for 'PLCopen XML serialization' with columns File, Status, and Code generation. Below the table are buttons for 'Generate PLCopen XML' and 'Download PLCopen XML'. Similarly, there is a 'PyUAF serialization' section with its own table and buttons for 'Generate pyUAF code' and 'Download pyUAF code'. At the bottom, a 'File contents:' section shows a code editor with XML code for the mtcs\_cover ontology.

OntoManager @ Mercator Telescope

Ontologies Dataset Problems Browse Query Systems Mechanics Electronics Software

Library **mtcs\_cover**

PLCopen XML serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml	File has been read	Not running

Generate PLCopen XML Download PLCopen XML

PyUAF serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py	File has been read	Not running

Generate pyUAF code Download pyUAF code

File contents:

Show PLCopen XML code Show PyUAF code

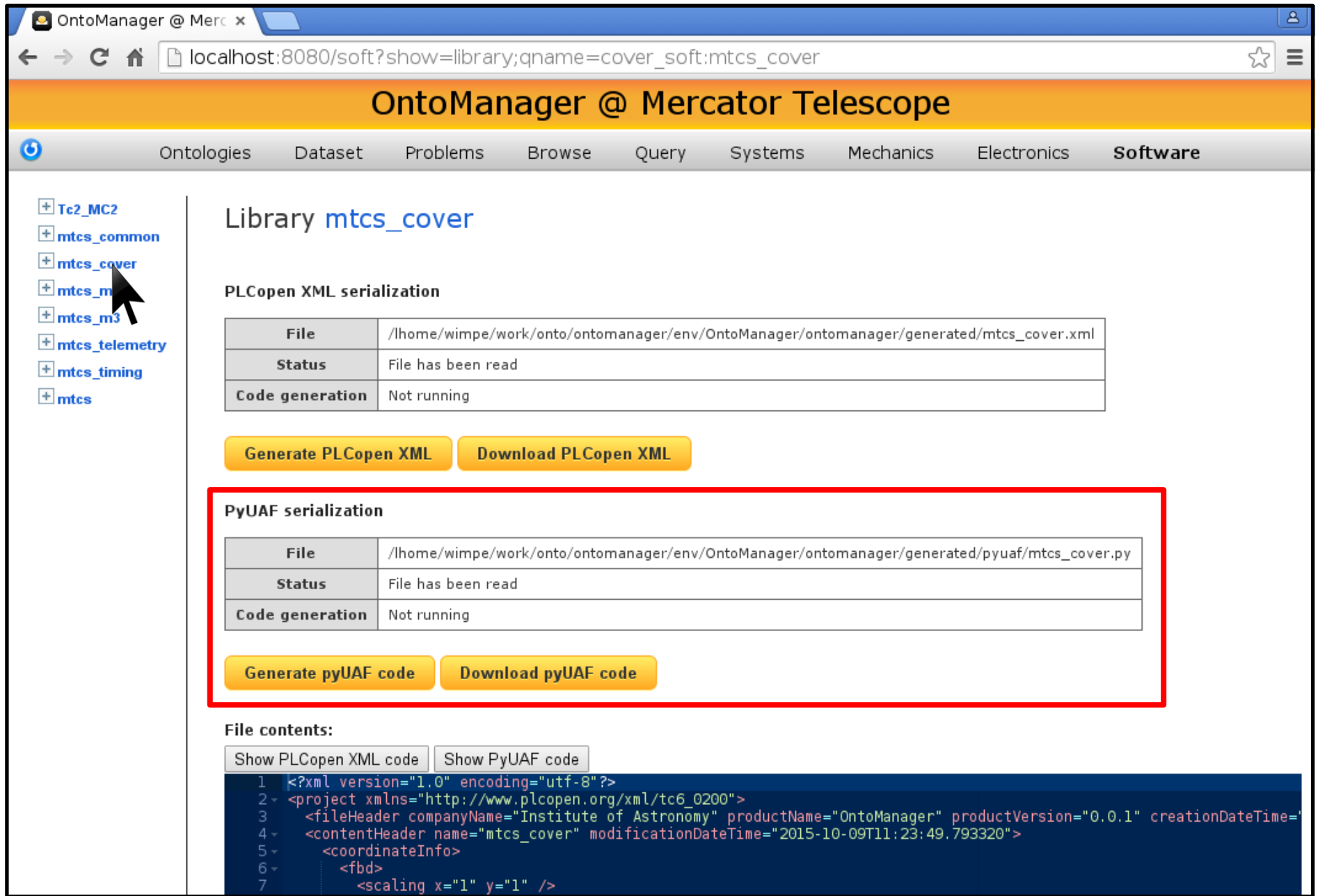
```
1 <?xml version="1.0" encoding="utf-8"?>
2 <project xmlns="http://www.plcopen.org/xml/tc6_0200">
3   <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
4   <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
5     <coordinateInfo>
6       <fbid>
7       <scaling x="1" y="1" />
```



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design



The screenshot shows the OntoManager @ Mercator Telescope web interface. The browser address bar shows the URL `localhost:8080/soft?show=library;qname=cover_soft:mtcs_cover`. The page title is "OntoManager @ Mercator Telescope". The navigation bar includes links for Ontologies, Dataset, Problems, Browse, Query, Systems, Mechanics, Electronics, and Software. The left sidebar shows a tree of ontologies, with `mtcs_cover` selected. The main content area displays the "Library mtcs\_cover" section, which includes a table for "PLCopen XML serialization" and a table for "PyUAF serialization". Both tables show the file path, status, and code generation status. Below the tables are buttons for "Generate PLCopen XML", "Download PLCopen XML", "Generate pyUAF code", and "Download pyUAF code". The PyUAF section is highlighted with a red border. At the bottom, there is a "File contents:" section with buttons for "Show PLCopen XML code" and "Show PyUAF code". The PyUAF code is displayed in a dark blue box.

OntoManager @ Mercator Telescope

Library `mtcs_cover`

PLCopen XML serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/mtcs_cover.xml	File has been read	Not running

Generate PLCopen XML Download PLCopen XML

PyUAF serialization

File	Status	Code generation
/lhome/wimpe/work/onto/ontomanager/env/OntoManager/ontomanager/generated/pyuaf/mtcs_cover.py	File has been read	Not running

Generate pyUAF code Download pyUAF code

File contents:

Show PLCopen XML code Show PyUAF code

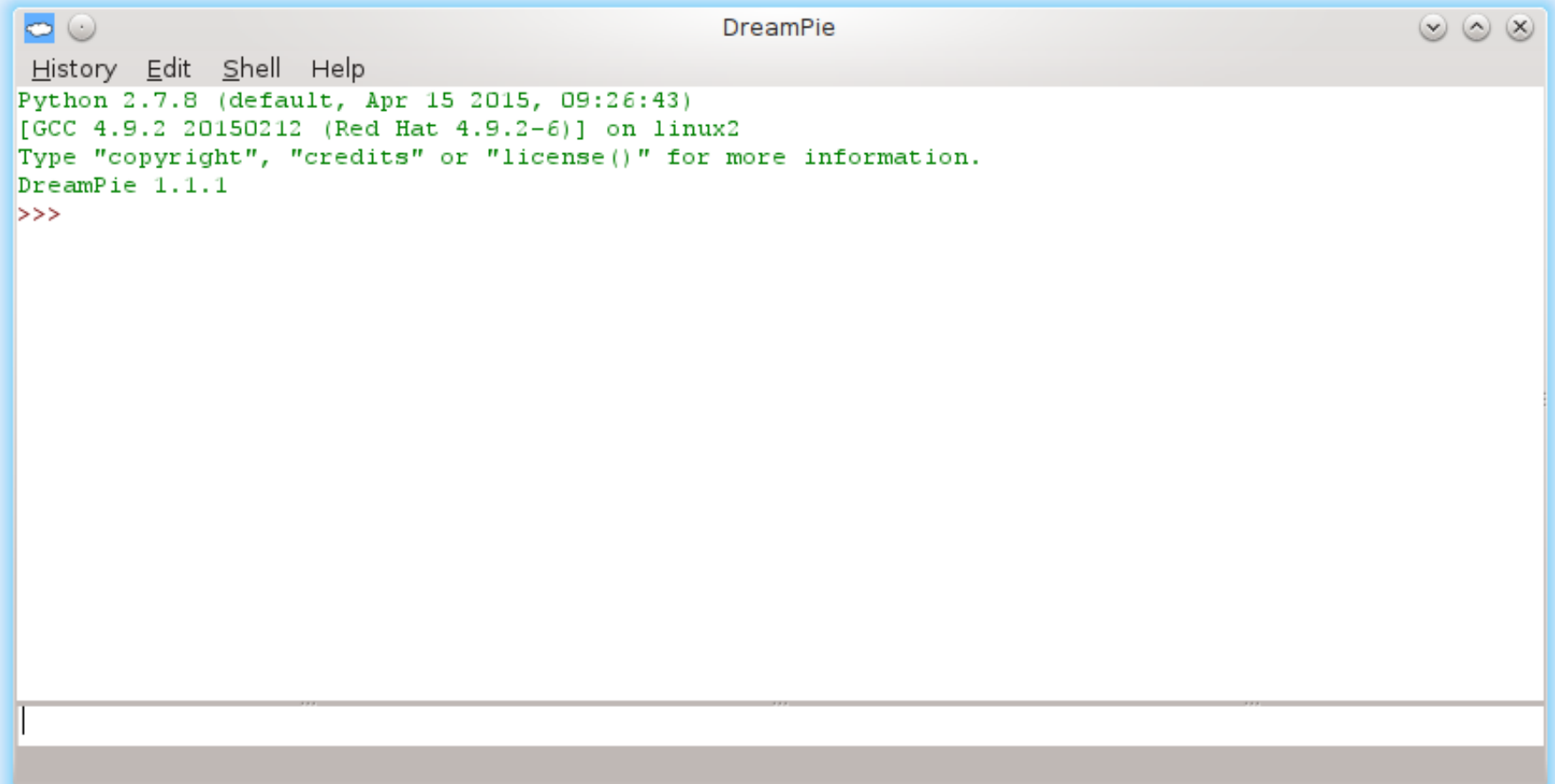
```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <project xmlns="http://www.plcopen.org/xml/tc6_0200">
3   <fileHeader companyName="Institute of Astronomy" productName="OntoManager" productVersion="0.0.1" creationDateTime="
4   <contentHeader name="mtcs_cover" modificationDateTime="2015-10-09T11:23:49.793320">
5     <coordinateInfo>
6       <fbid>
7       <scaling x="1" y="1" />
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



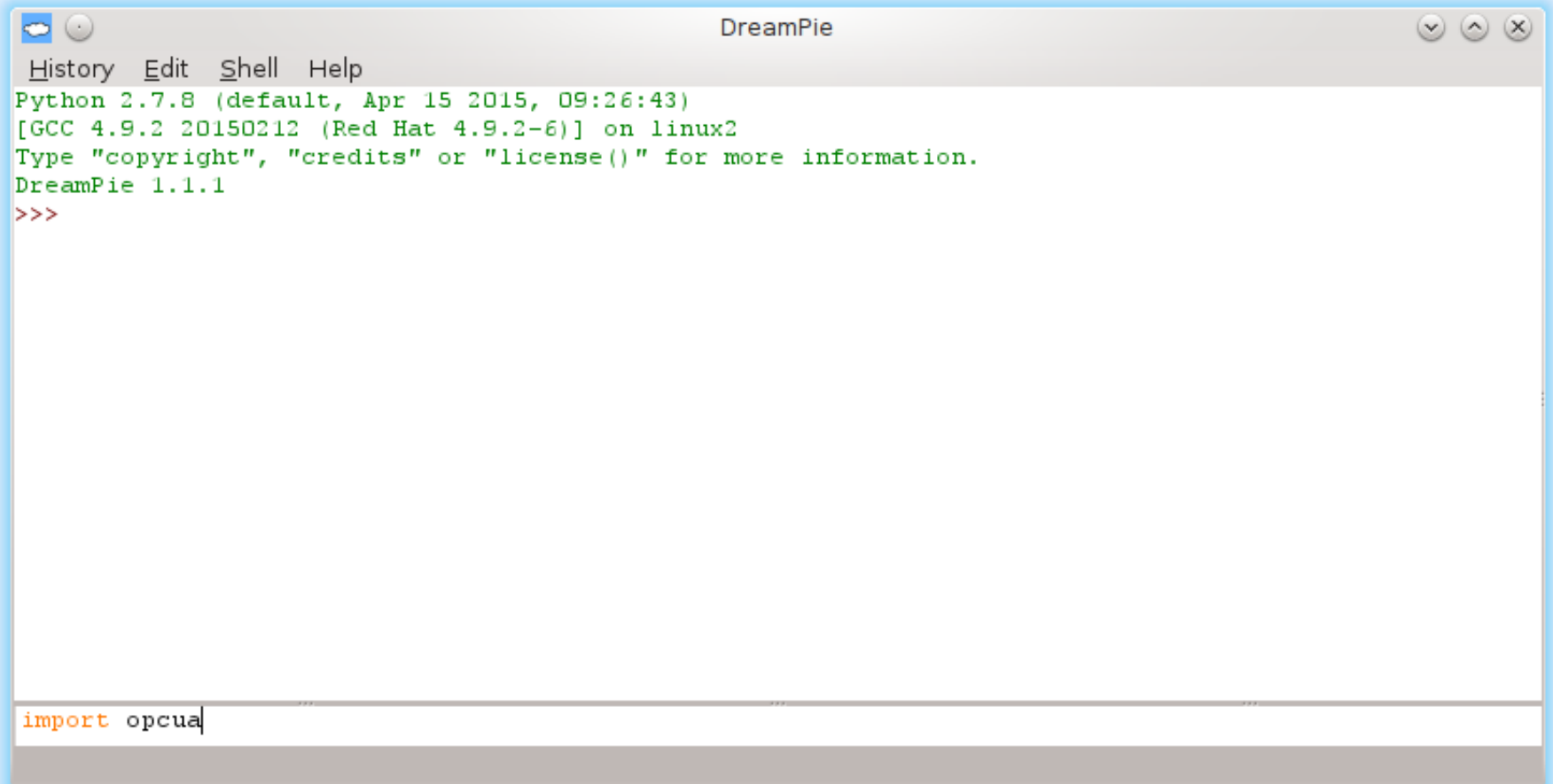
The screenshot shows a window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The main area displays the following text in green: "Python 2.7.8 (default, Apr 15 2015, 09:26:43)", "[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2", "Type \"copyright\", \"credits\" or \"license()\" for more information.", and "DreamPie 1.1.1". At the bottom, there is a red prompt ">>>" indicating a ready state for input.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>>

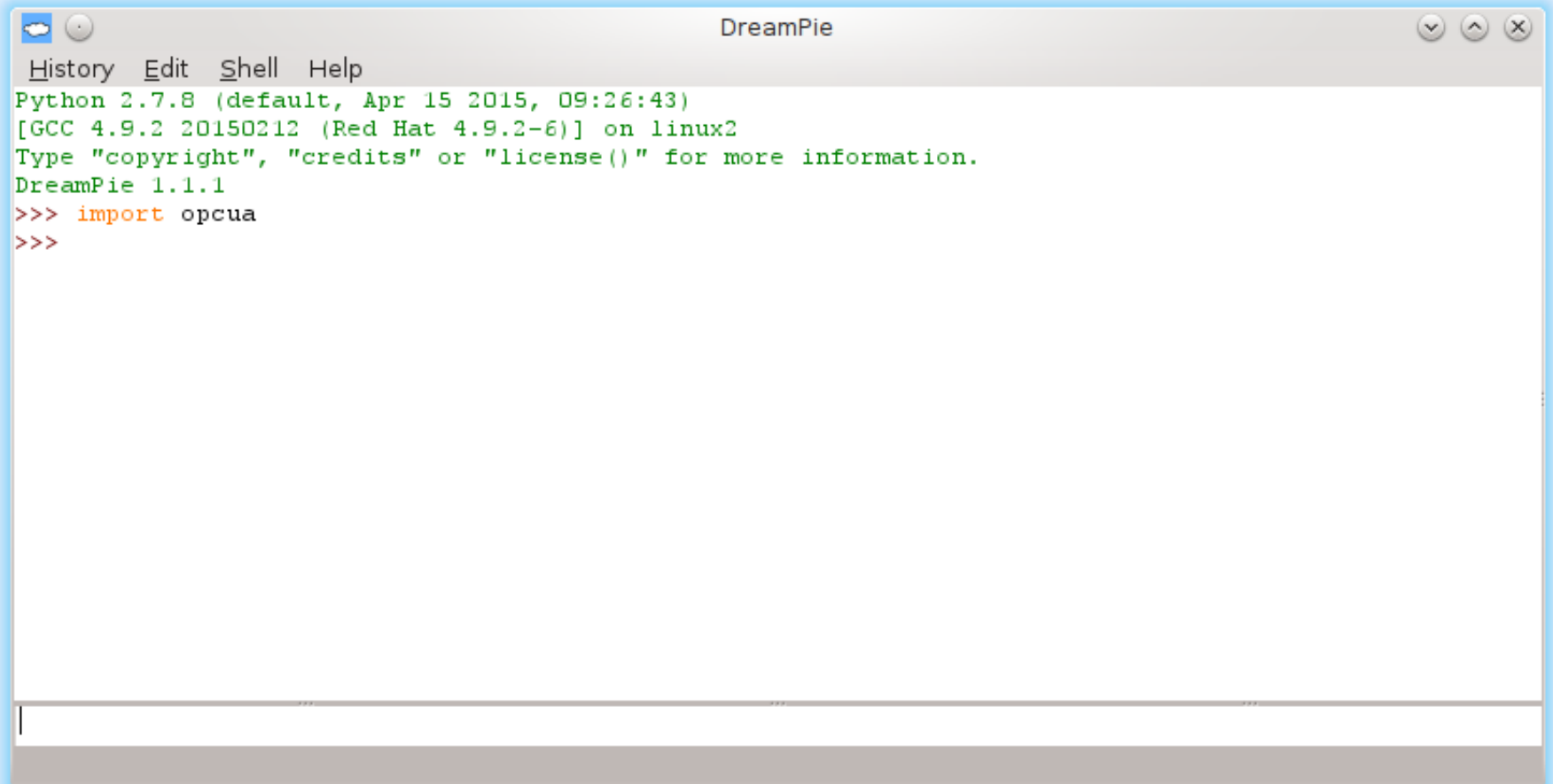
import opcua
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>>
```

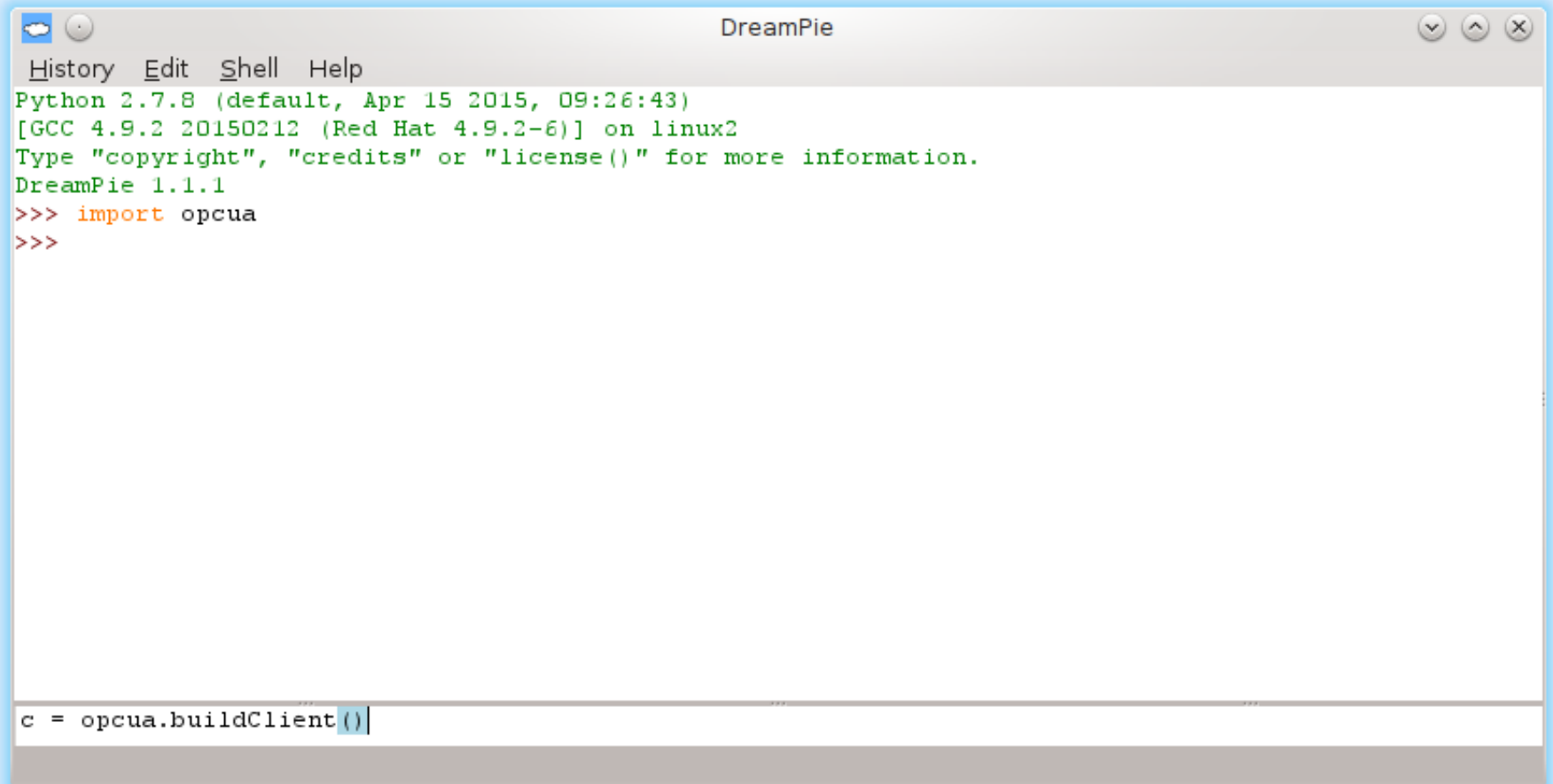


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output displays the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the command `>>> import opcua` and is now at the prompt `>>>`. At the bottom of the window, the code `c = opcua.buildClient({})` is visible, with the opening curly brace highlighted.

```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>>

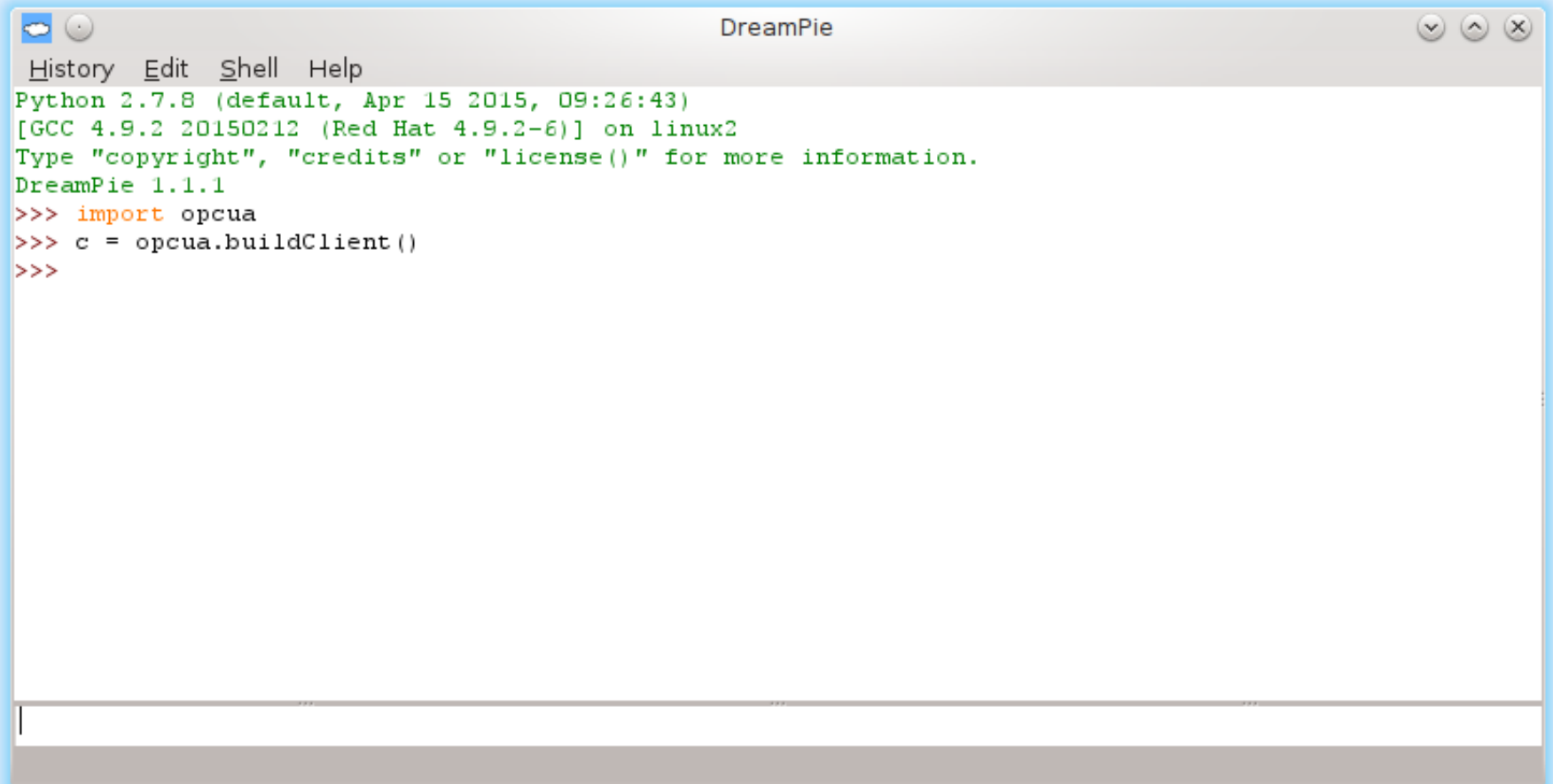
c = opcua.buildClient({})
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The image shows a terminal window titled "DreamPie". The window has a menu bar with "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered three lines of Python code: `>>> import opcua`, `>>> c = opcua.buildClient()`, and `>>>`.

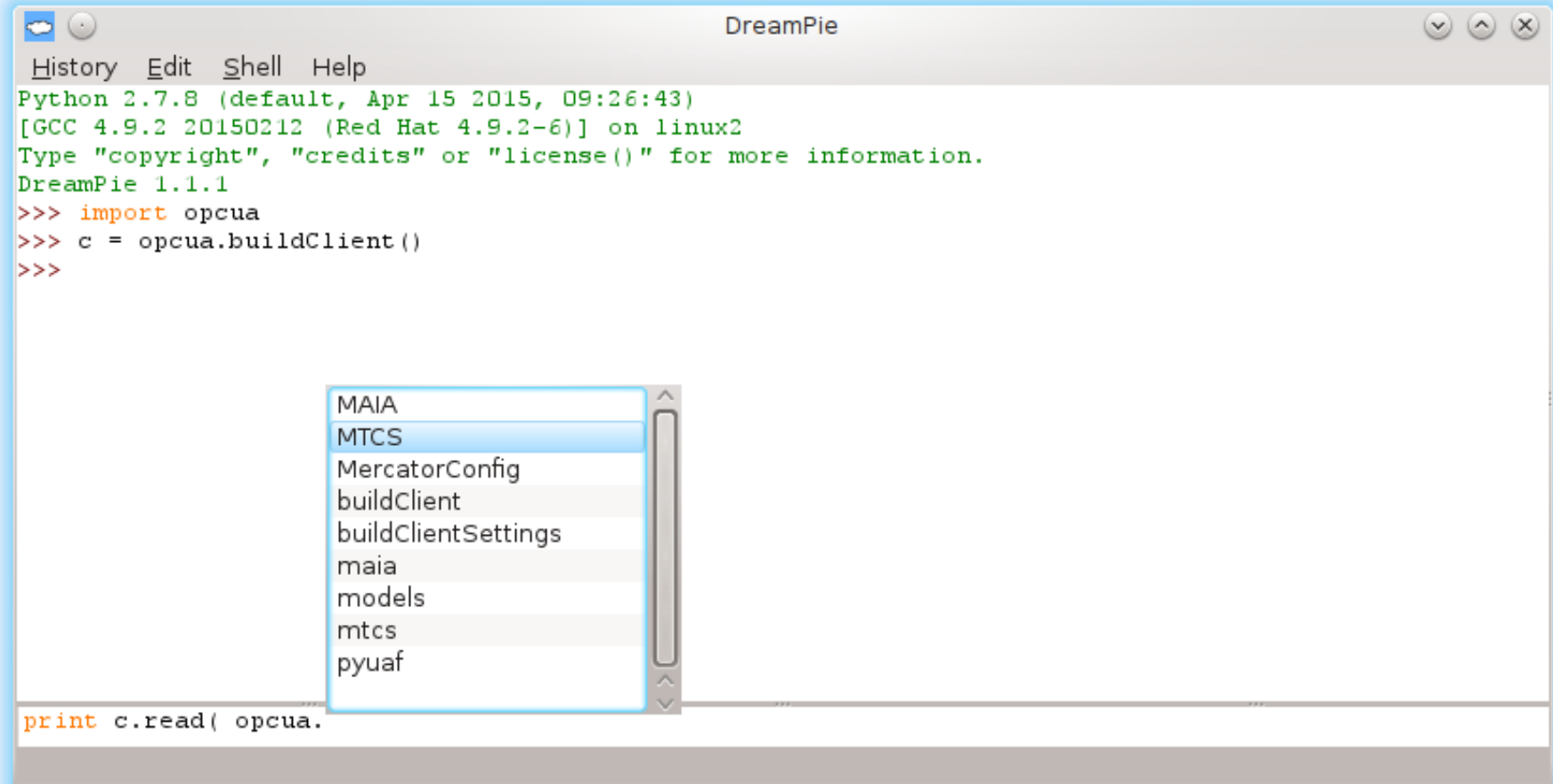
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The main area is a Python 2.7.8 shell. The text in the shell is as follows:

```
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

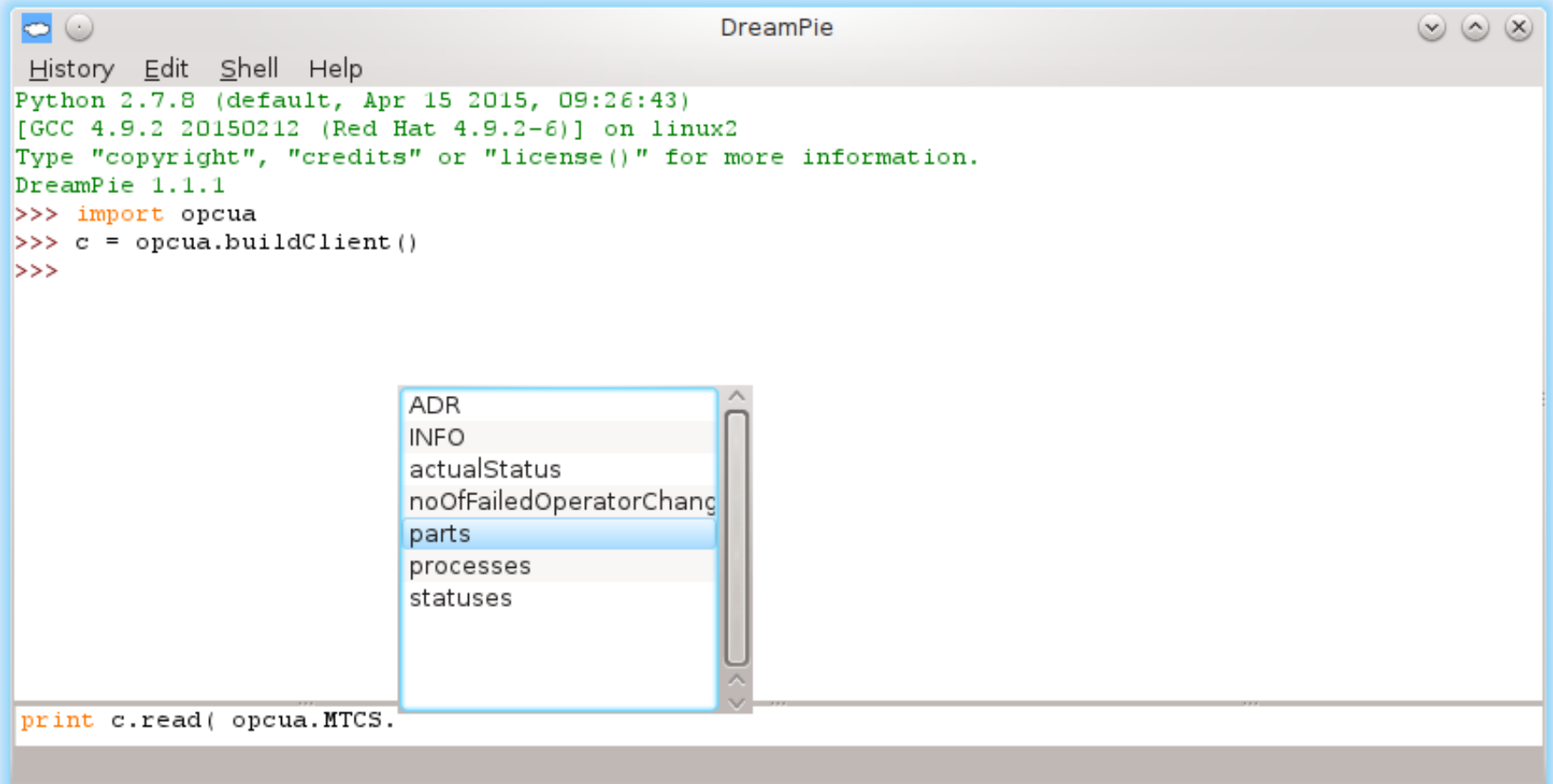
A dropdown menu is open, showing a list of items: MAIA, MTCS, MercatorConfig, buildClient, buildClientSettings, maia, models, mtcs, and pyuaf. The "MTCS" item is currently selected. At the bottom of the shell, the text `print c.read( opcua.` is visible.

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, showing the following options: ADR, INFO, actualStatus, noOfFailedOperatorChang, parts (highlighted), processes, and statuses.

At the bottom of the terminal, the following code is visible:

```
print c.read( opcua.MTCS.
```

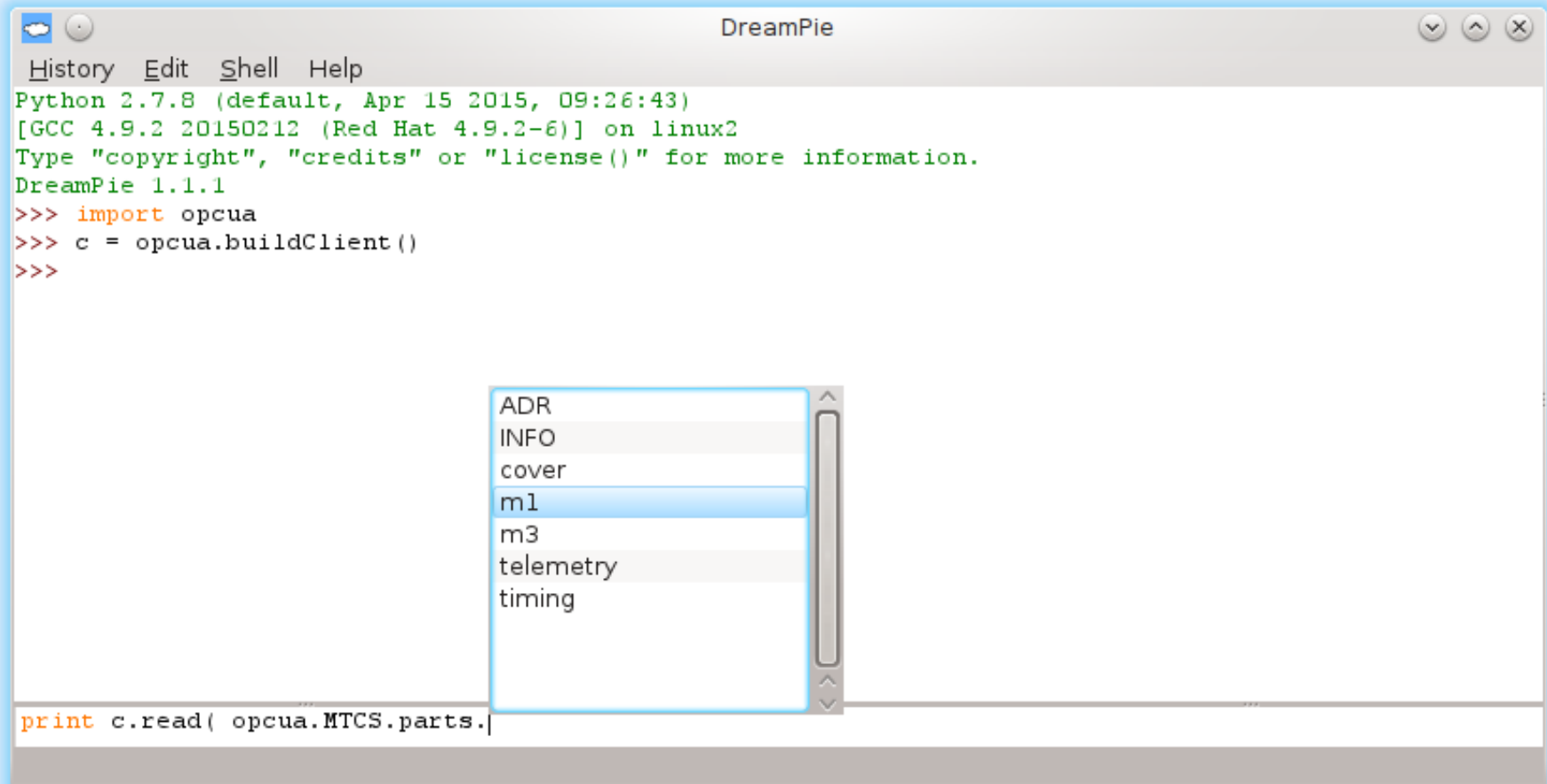


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following Python code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, showing a list of items: "ADR", "INFO", "cover", "m1", "m3", "telemetry", and "timing". The item "m1" is currently selected. At the bottom of the terminal, the following code is partially visible:

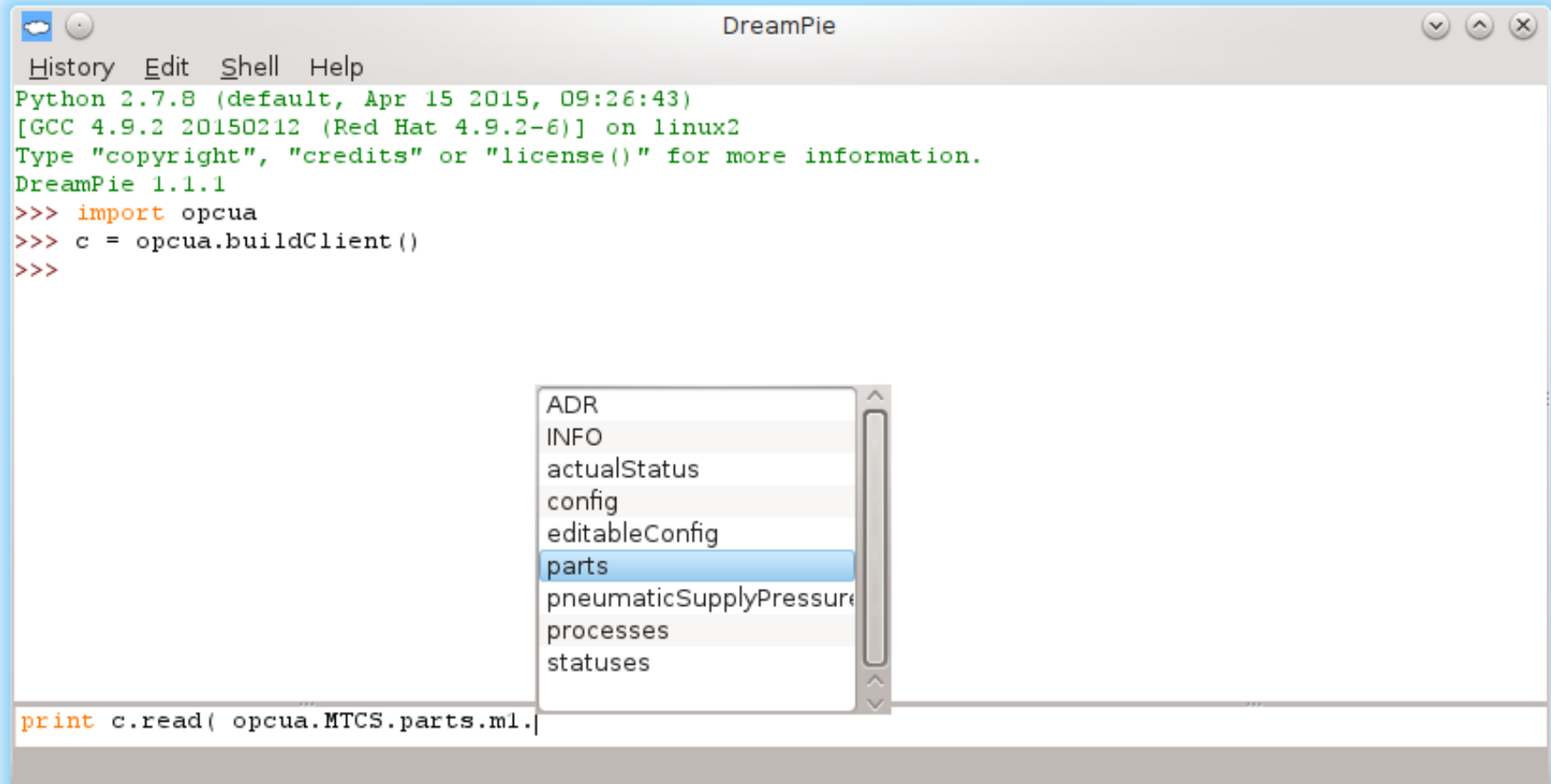
```
print c.read( opcua.MTCS.parts,|
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python environment (Python 2.7.8, GCC 4.9.2) and the execution of the following code:

```
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, displaying a list of attributes: ADR, INFO, actualStatus, config, editableConfig, **parts** (highlighted), pneumaticSupplyPressure, processes, and statuses.

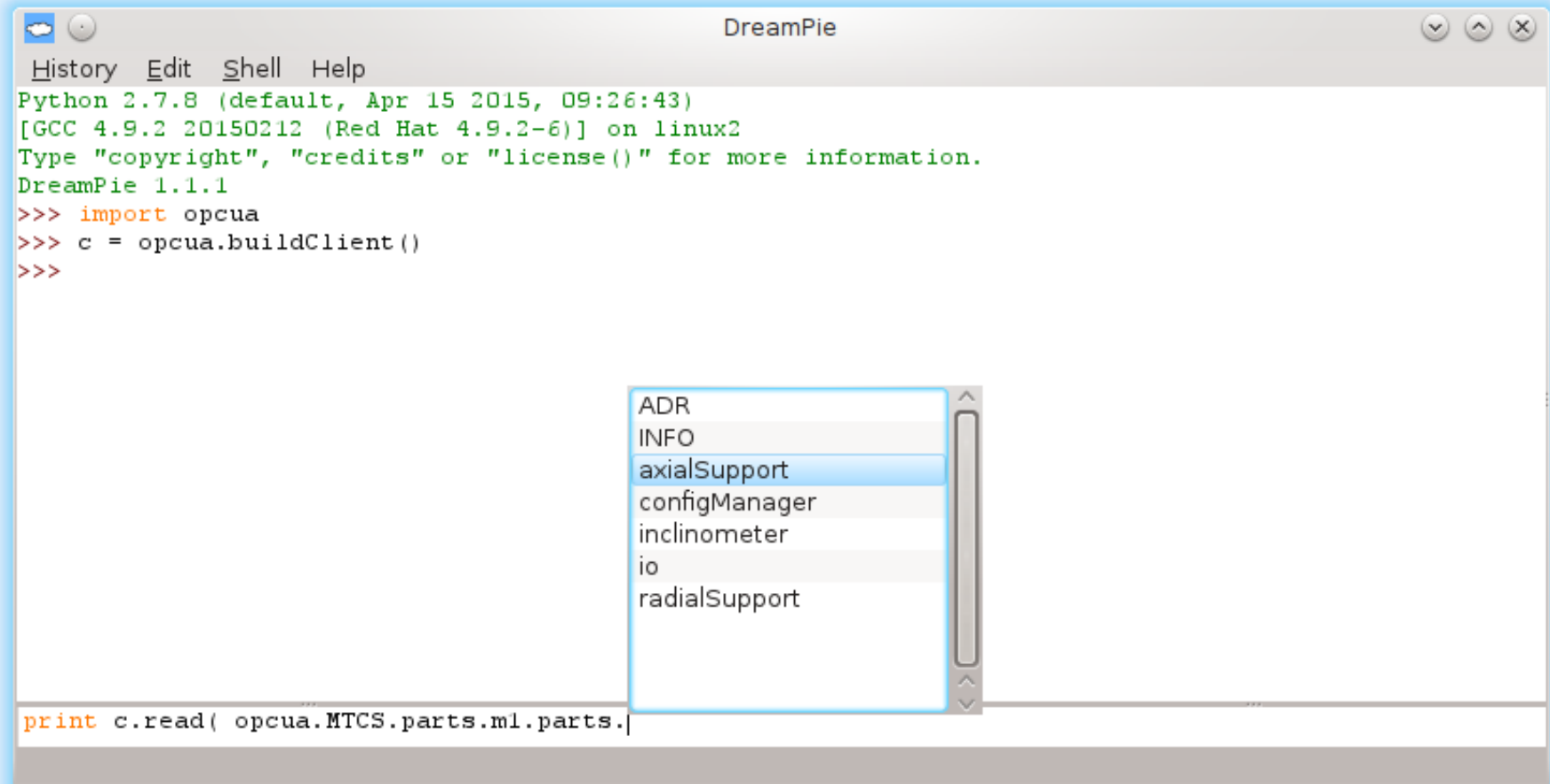
```
print c.read( opcua.MTCS.parts.ml.
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a window titled "DreamPie" with a menu bar (History, Edit, Shell, Help) and a terminal-like interface. The terminal displays the following text:

```
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open, showing a list of items: ADR, INFO, axialSupport (highlighted), configManager, inclinometer, io, and radialSupport. At the bottom of the window, the following code is partially visible:

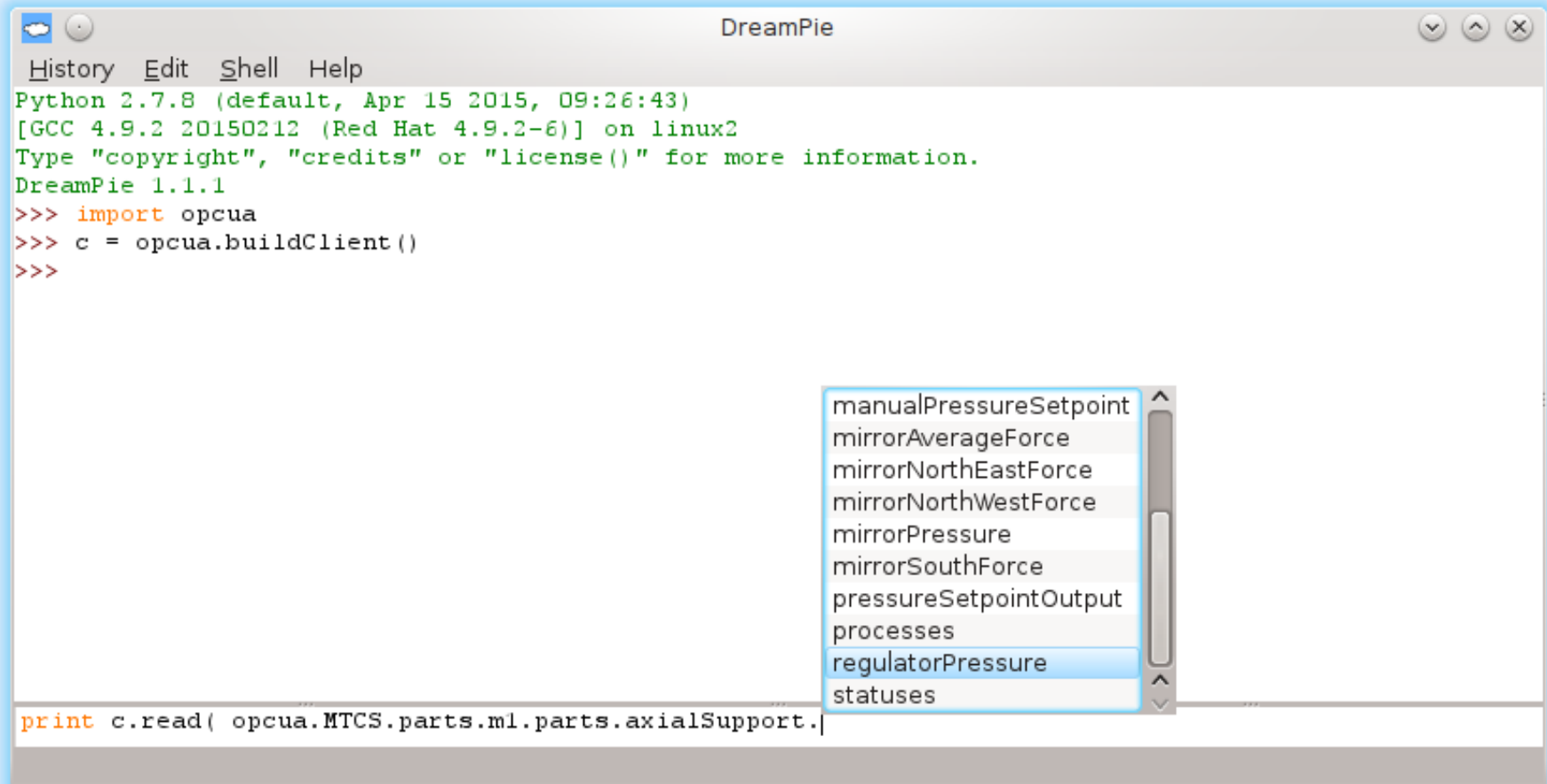
```
print c.read( opcua.MTCS.parts.ml.parts.)
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

manualPressureSetpoint
mirrorAverageForce
mirrorNorthEastForce
mirrorNorthWestForce
mirrorPressure
mirrorSouthForce
pressureSetpointOutput
processes
regulatorPressure
statuses

print c.read( opcua.MTCS.parts.ml.parts.axialSupport.
```

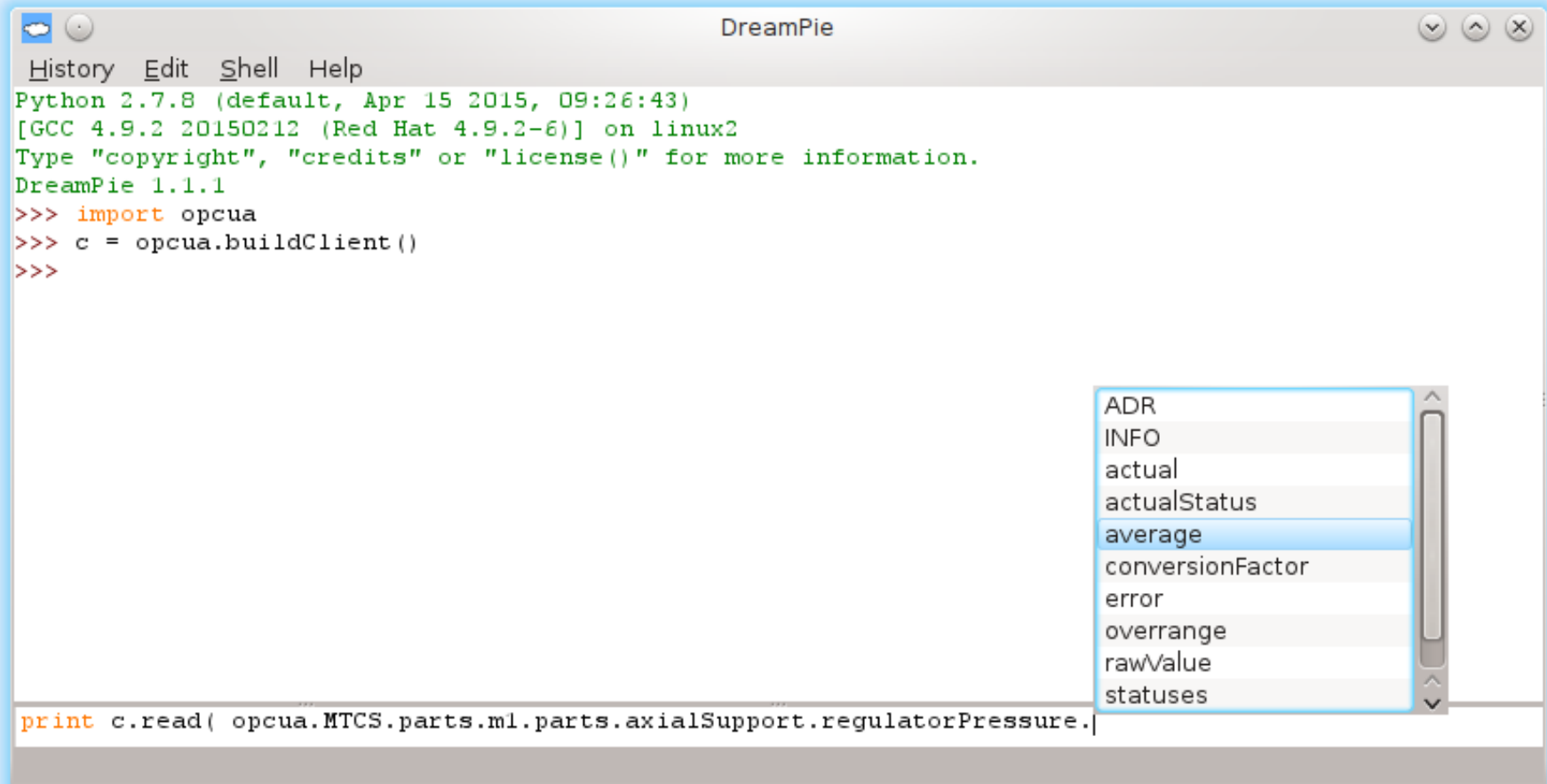


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



The screenshot shows a terminal window titled "DreamPie" with a menu bar containing "History", "Edit", "Shell", and "Help". The terminal output shows the Python version (2.7.8), GCC version (4.9.2), and the DreamPie version (1.1.1). The user has entered the following Python code:

```
>>> import opcua
>>> c = opcua.buildClient()
>>>
```

A dropdown menu is open on the right side of the terminal, listing the following attributes: ADR, INFO, actual, actualStatus, **average** (highlighted), conversionFactor, error, overrange, rawValue, and statuses. The terminal prompt is currently at the end of the line:

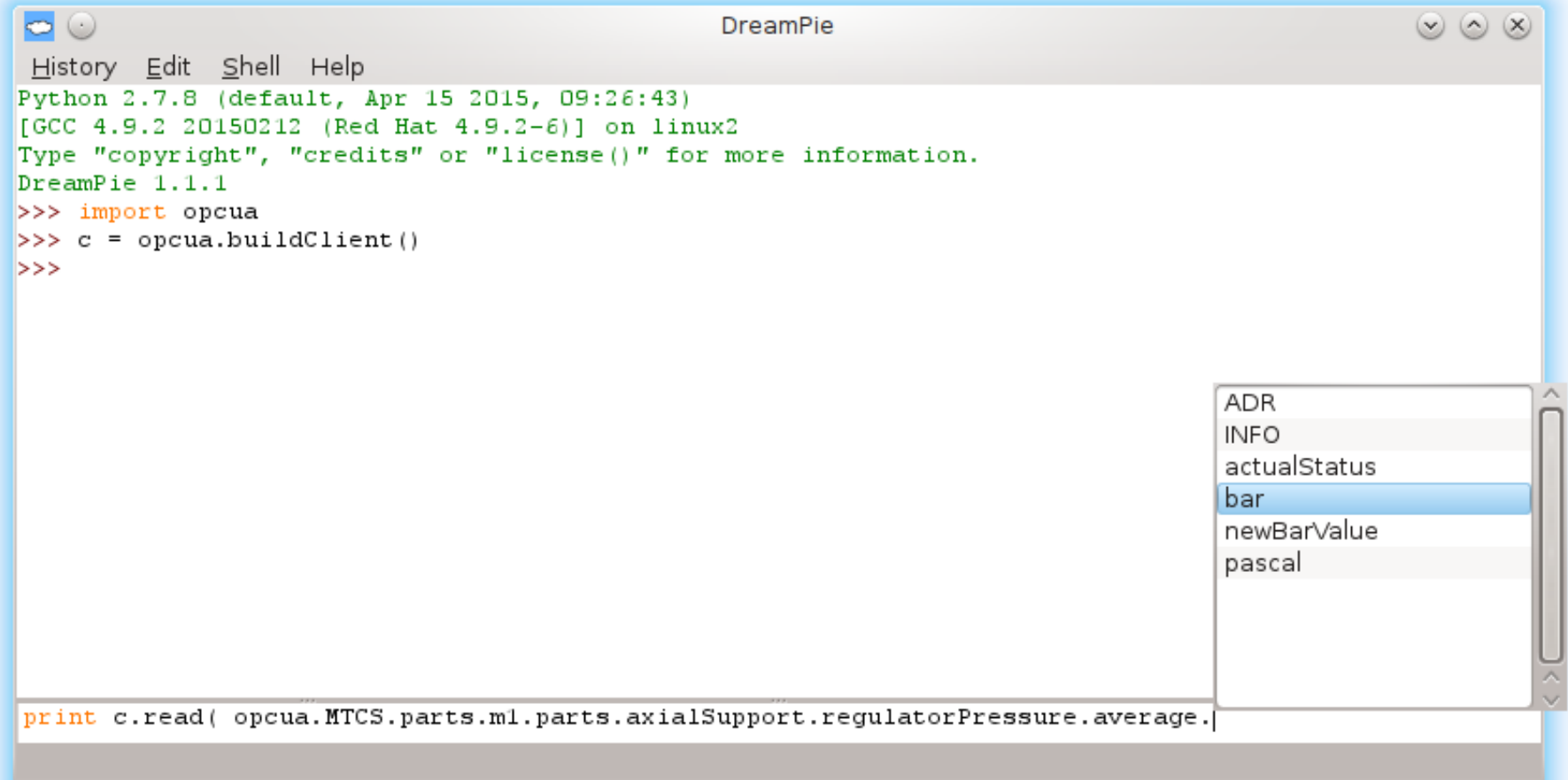
```
print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.)
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

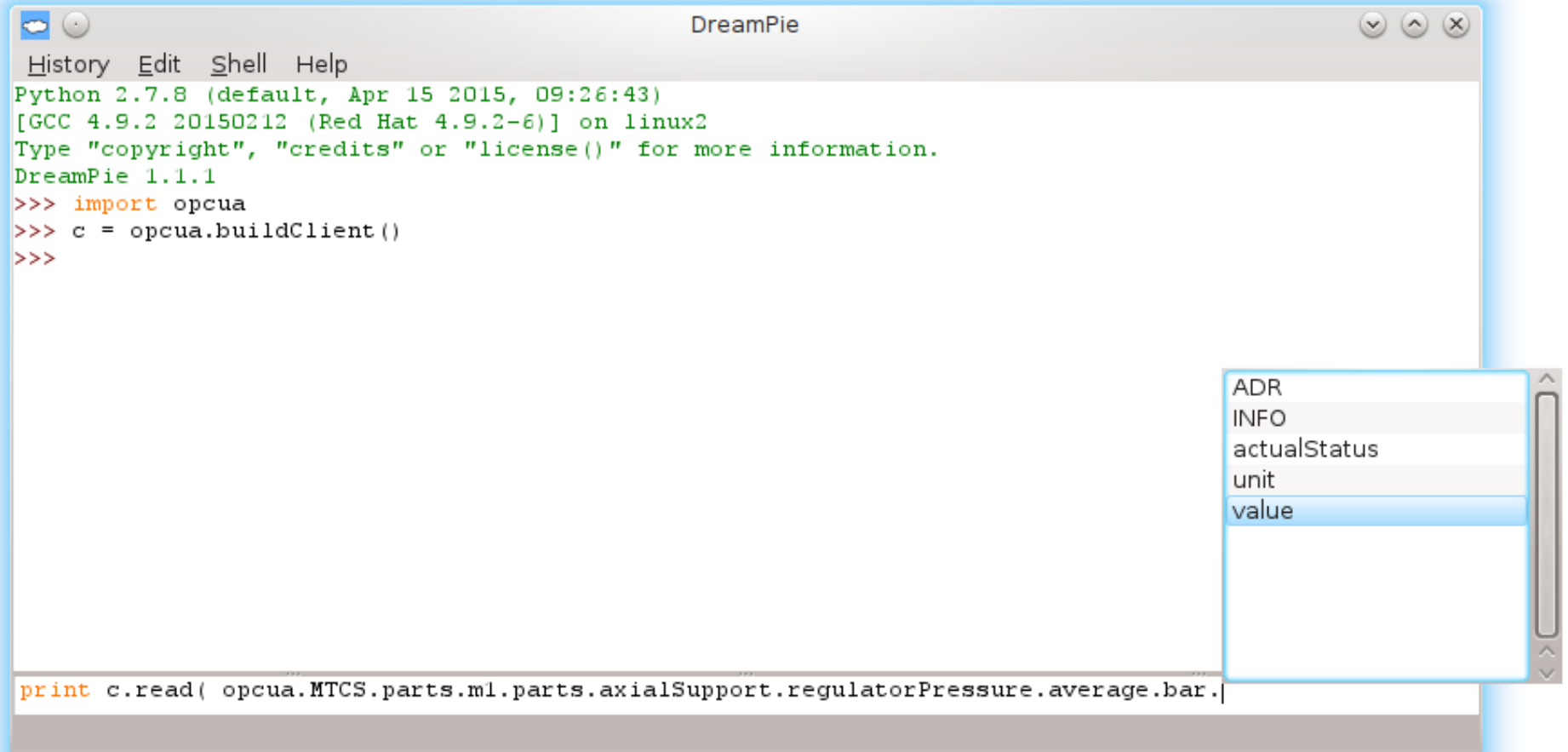
print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.average,
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.average.bar.

```

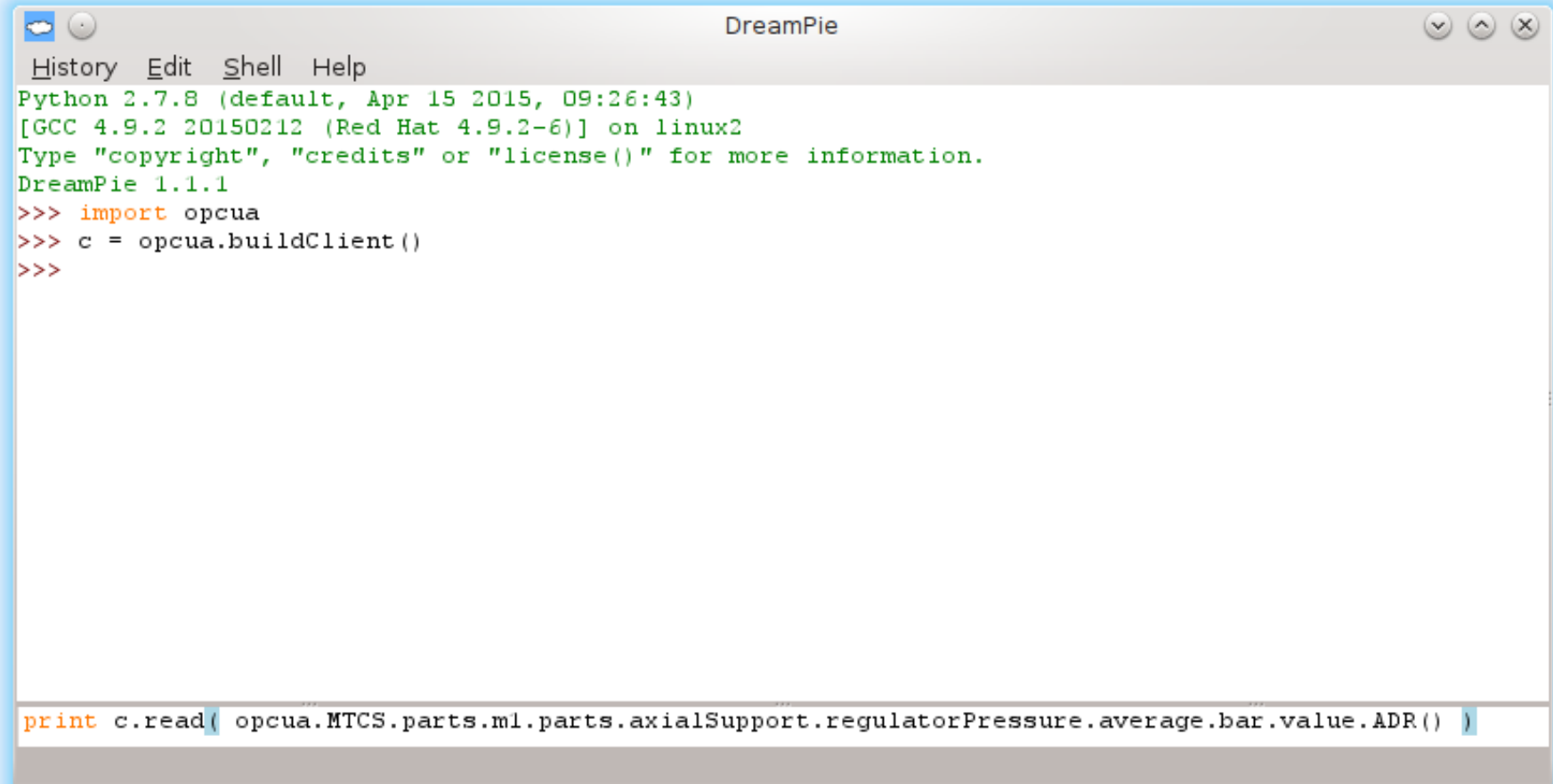
ADR  
INFO  
actualStatus  
unit  
value

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>>

print c.read( opcua.MTCS.parts.ml.parts.axialSupport.regulatorPressure.average.bar.value.ADR() )
```

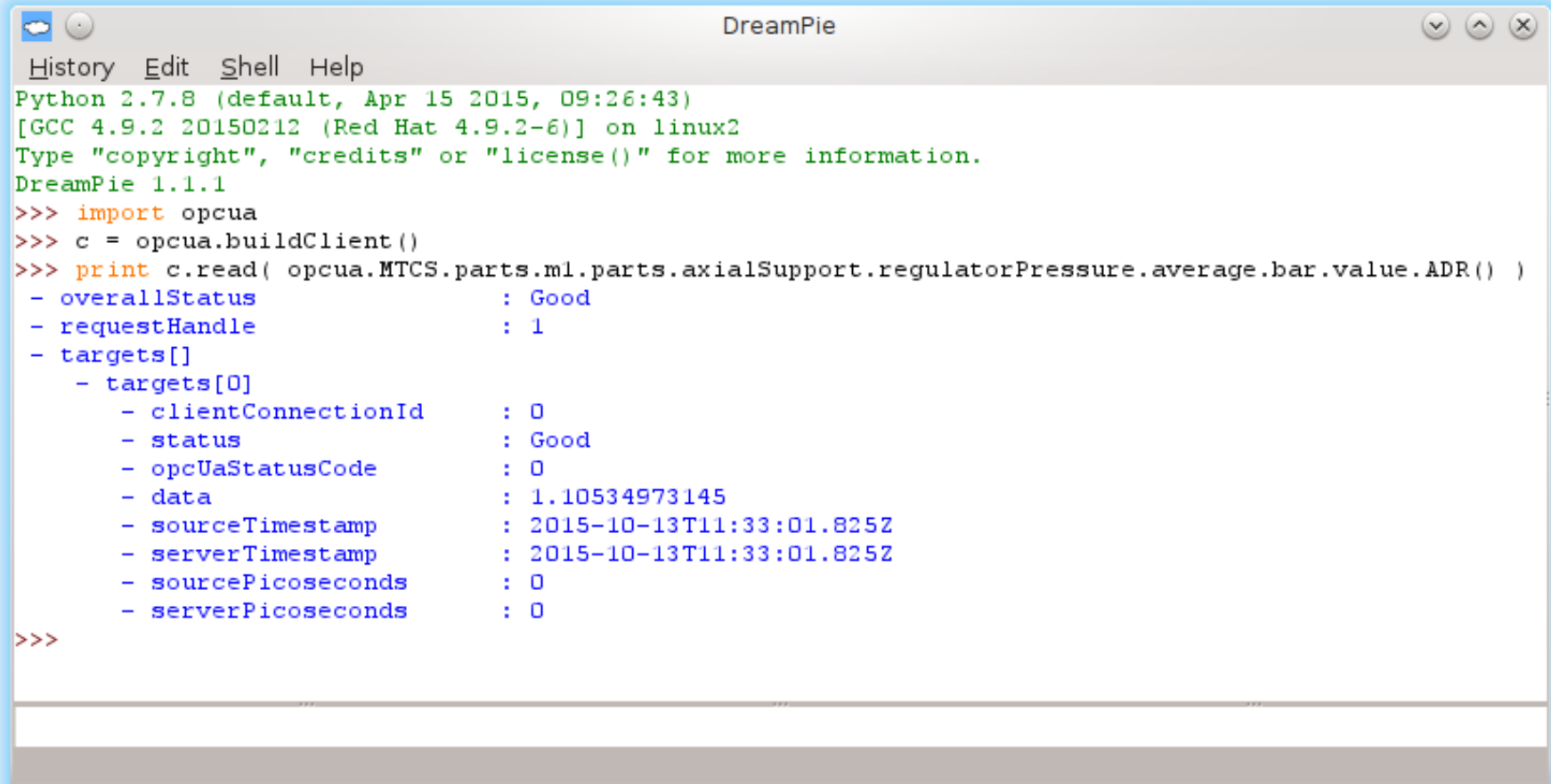


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



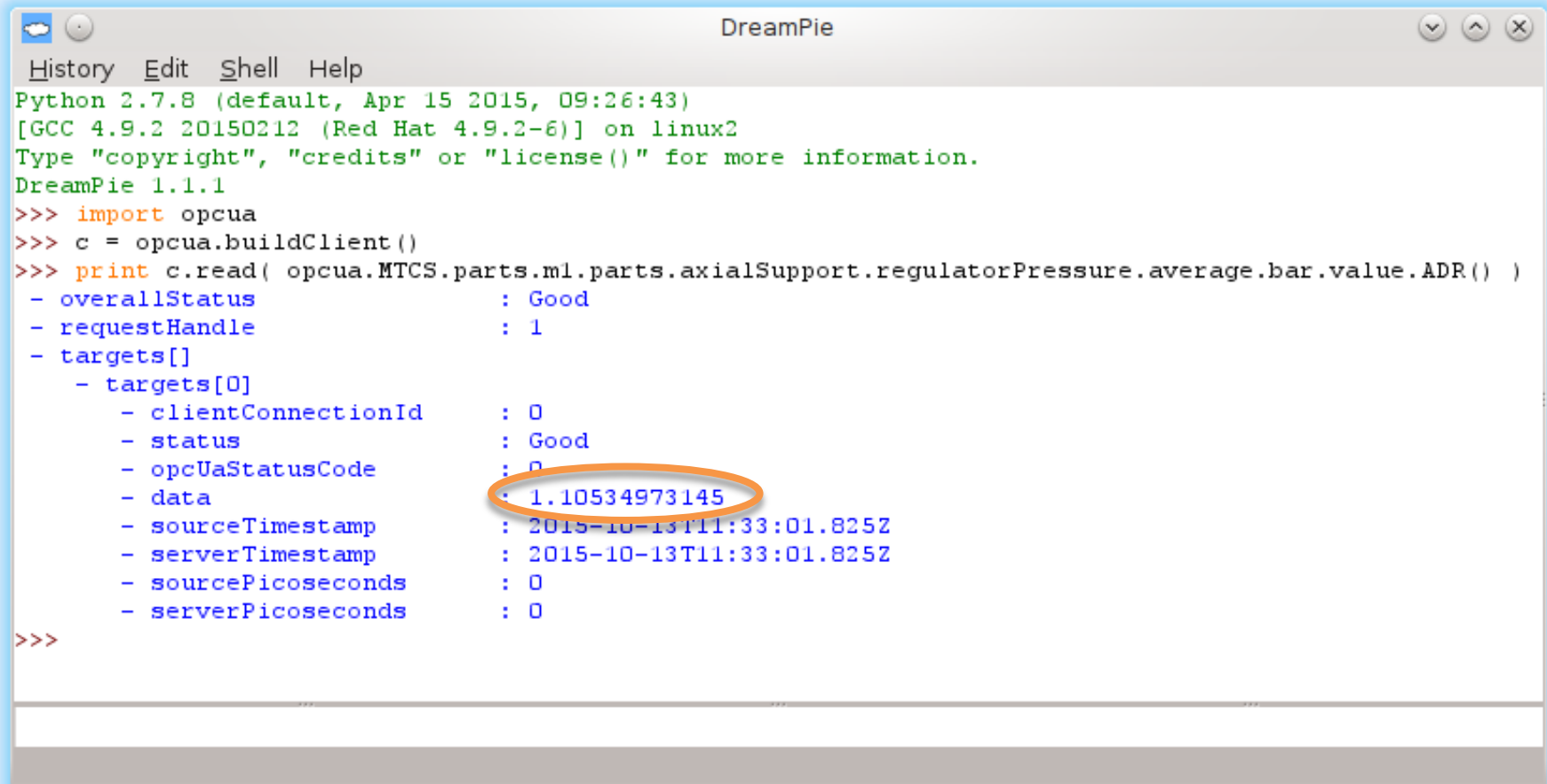
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>> print c.read( opcua.MTCS.parts.m1.parts.axialSupport.regulatorPressure.average.bar.value.ADR() )
- overallStatus          : Good
- requestHandle          : 1
- targets[]
  - targets[0]
    - clientConnectionId  : 0
    - status              : Good
    - opcUaStatusCode     : 0
    - data                : 1.10534973145
    - sourceTimestamp     : 2015-10-13T11:33:01.825Z
    - serverTimestamp     : 2015-10-13T11:33:01.825Z
    - sourcePicoSeconds   : 0
    - serverPicoSeconds   : 0
>>>
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

# Software design

- Generated Python code (client side)
  - Based on our OPC UA library “UAF”: <http://github.com/uaf/uaf>



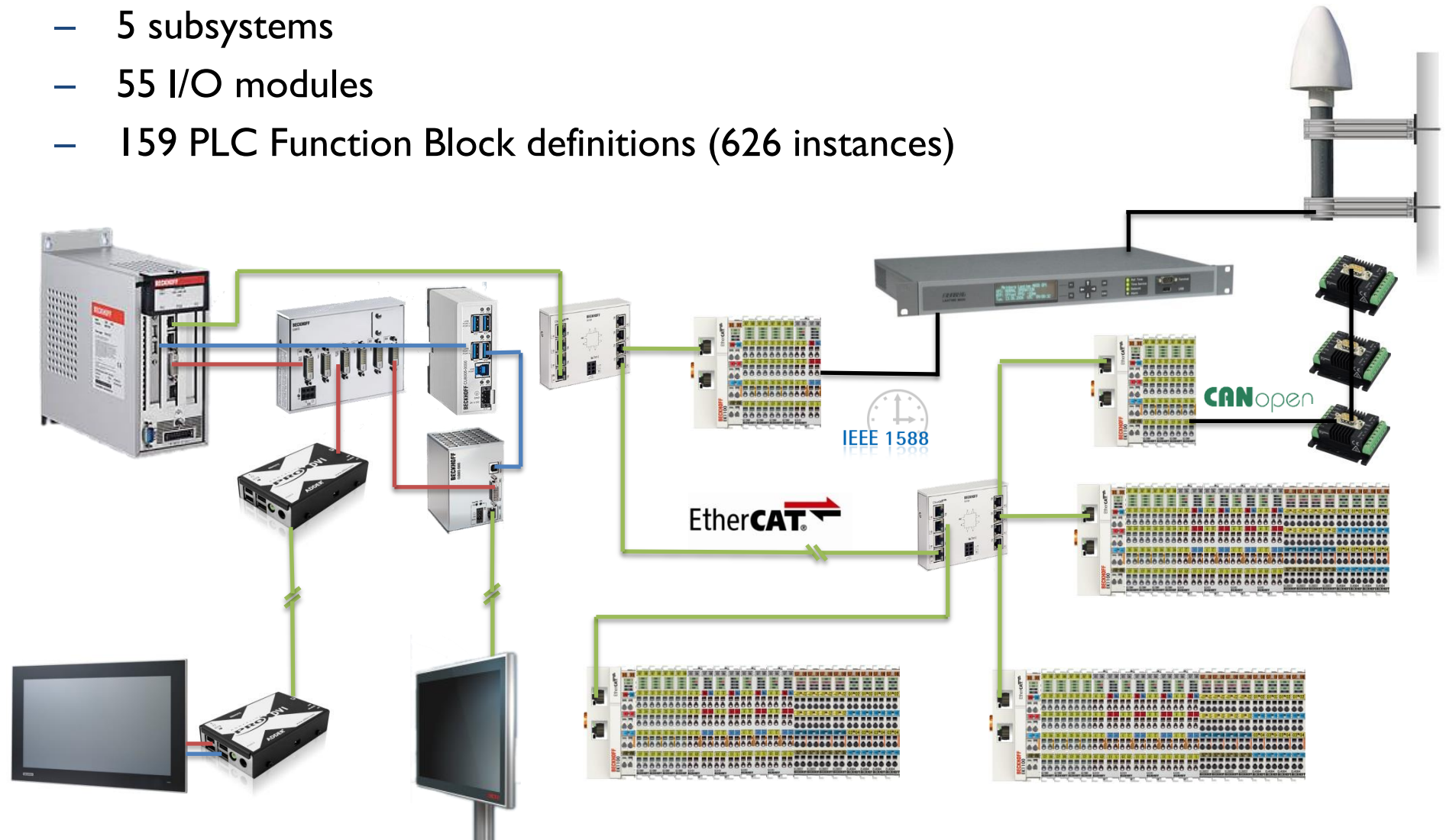
```
DreamPie
History Edit Shell Help
Python 2.7.8 (default, Apr 15 2015, 09:26:43)
[GCC 4.9.2 20150212 (Red Hat 4.9.2-6)] on linux2
Type "copyright", "credits" or "license()" for more information.
DreamPie 1.1.1
>>> import opcua
>>> c = opcua.buildClient()
>>> print c.read( opcua.MTCS.parts.m1.parts.axialSupport.regulatorPressure.average.bar.value.ADR() )
- overallStatus          : Good
- requestHandle          : 1
- targets[]
  - targets[0]
    - clientConnectionId  : 0
    - status              : Good
    - opcUaStatusCode     : 0
    - data                : 1.10534973145
    - sourceTimestamp     : 2015-10-13T11:33:01.825Z
    - serverTimestamp     : 2015-10-13T11:33:01.825Z
    - sourcePicoSeconds   : 0
    - serverPicoSeconds   : 0
>>>
```

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- Conclusions

## Results

- Currently in operation:
  - 1 PLC
  - 5 subsystems
  - 55 I/O modules
  - 159 PLC Function Block definitions (626 instances)





## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

## Results





## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

## Results



## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

## Results

- User Interface (HMI) running **on** the PLC

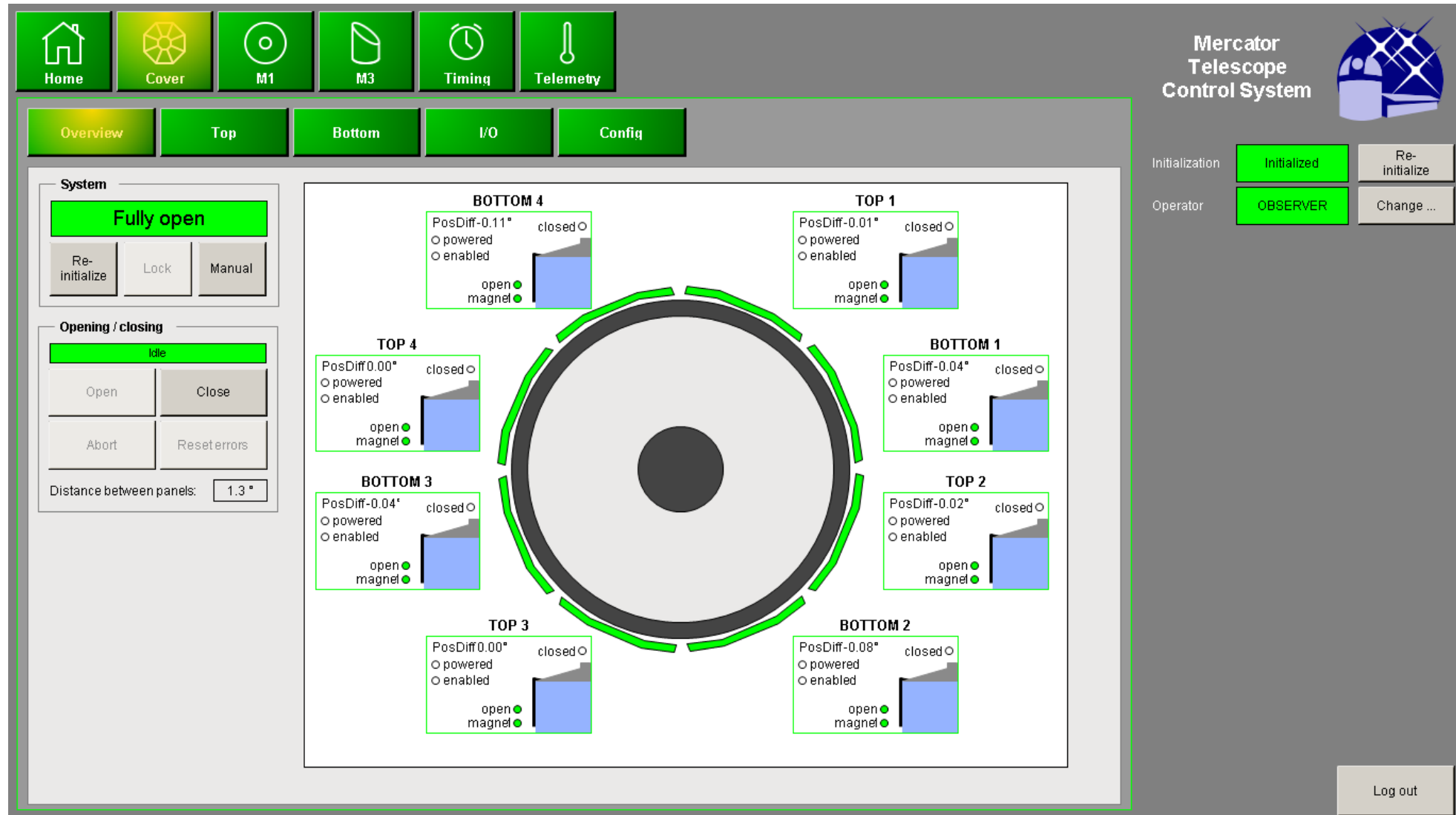


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

## Results

- User Interface (HMI) running **on** the PLC

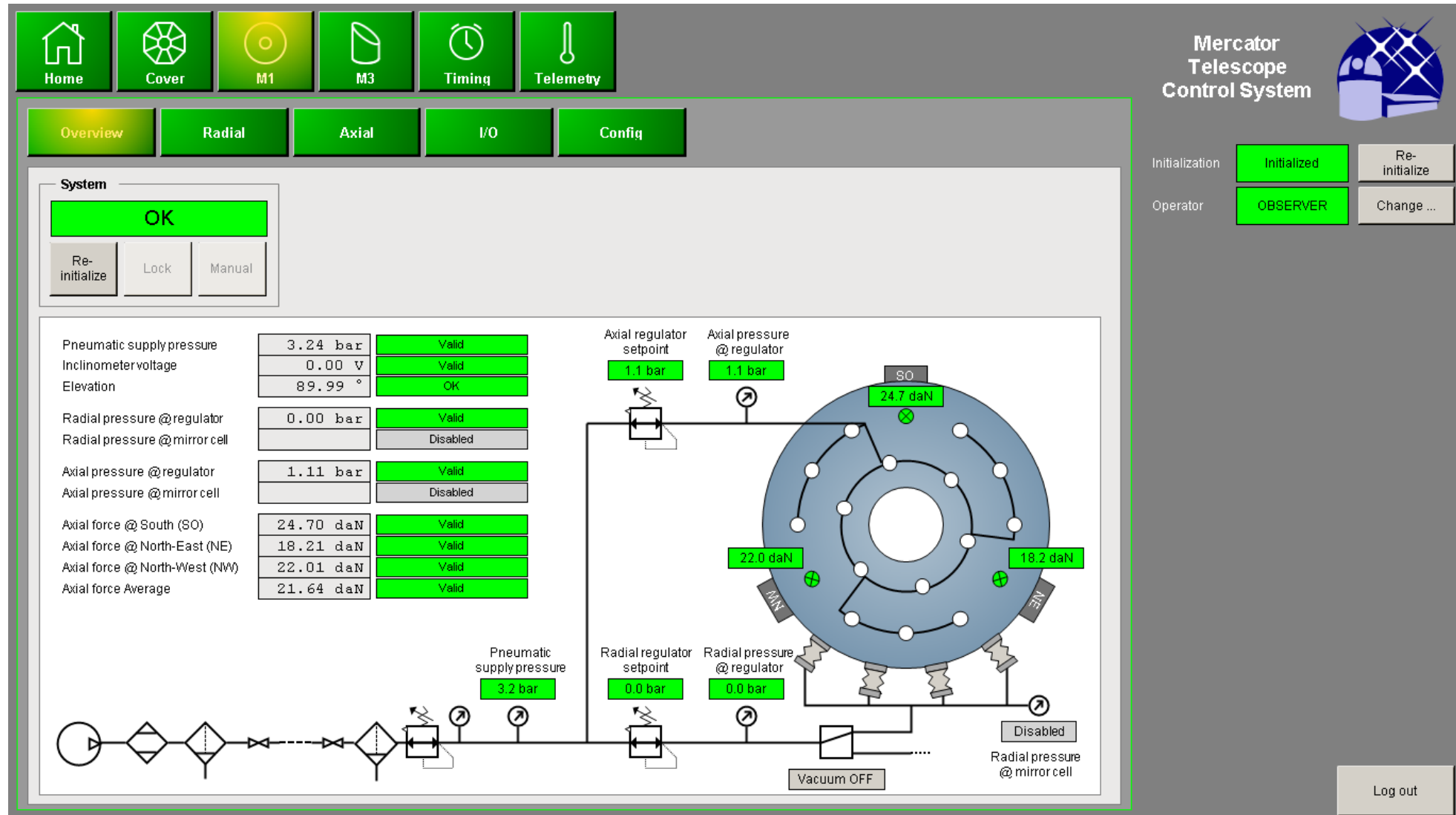


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- **How to use them?**
- Conclusions

## Results

- User Interface (HMI) running **on** the PLC





## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

# Conclusions

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

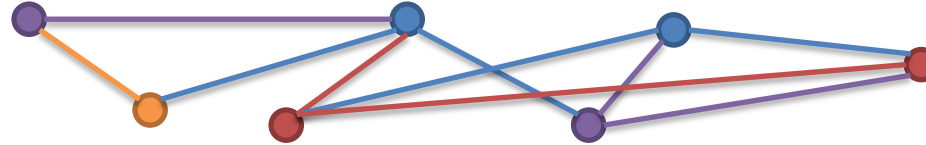
## So, why semantics matter?

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

## So, why semantics matter?

- I. Because every piece of information is just one query “away”



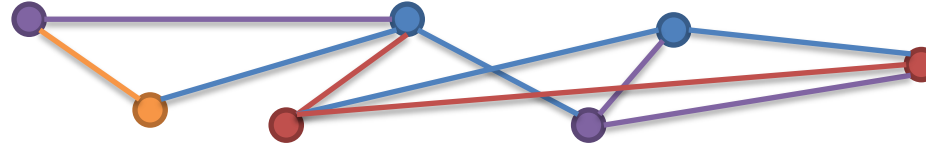
➔ **organize, integrate, browse, find (query) information**

## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

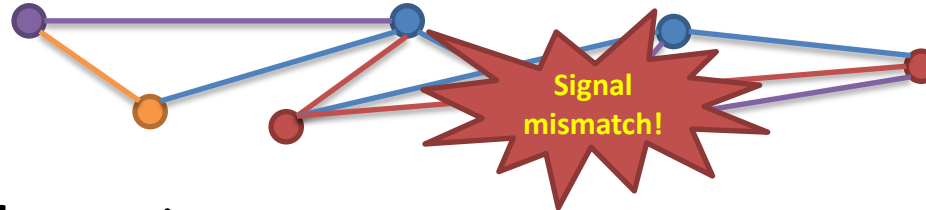
## So, why semantics matter?

1. Because every piece of information is just one query “away”



➔ **organize, integrate, browse, find (query)** information

2. Because well defined semantics allow model verification



➔ **verify** information

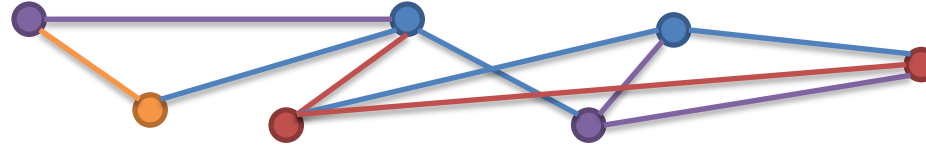


## Why semantics matter?

- What are semantic models?
- Where to apply them?
- How to apply them?
- How to build them?
- How to use them?
- **Conclusions**

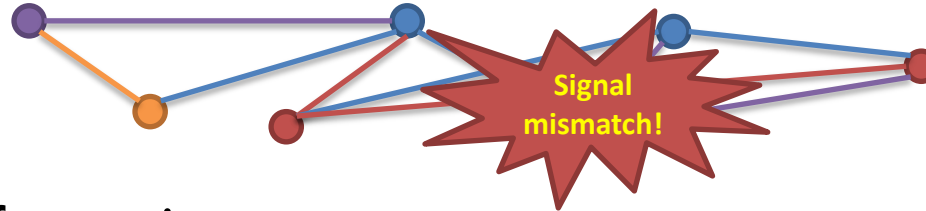
## So, why semantics matter?

1. Because every piece of information is just one query “away”



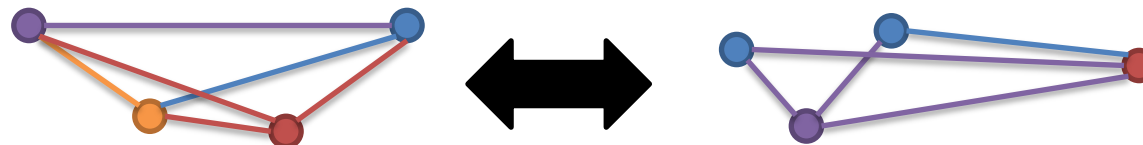
→ **organize, integrate, browse, find (query)** information

2. Because well defined semantics allow model verification



→ **verify** information

3. Because they're a key enabling technology for future “smart” systems



→ **share** information

A photograph of an astronomical observatory on a mountain peak at night. The sky is dark with many stars visible. A layer of clouds is illuminated from below, creating a warm, orange glow. The observatory building and its dome are silhouetted against the sky.

Thanks!

Any questions?

[wim.pessemier@ster.kuleuven.be](mailto:wim.pessemier@ster.kuleuven.be)